Oracle® Application Server

Performance and Tuning Guide

Release 4.0.8.1

September 1999

Part No. A60120-03

ORACLE®

Oracle Application Server Release 4.0.8.1 Performance and Tuning Guide

Part No. A60120-03

Copyright © 1996, 1999, Oracle Corporation. All rights reserved.

Authors: Sanjay Singh, Janice Nygard, Francisco Abedrabbo

Contributors: Sharon Malek, Bruce Irvin, Alice Chan, Carol Orange, Rachita Atal, Bryan Burns, Thomas Van Raalte, Kiri Burnovas, Miranda Padgett

The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the programs.

The programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these programs, no part of these programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle is a registered trademark, and the Oracle logo, NLS*WorkBench, Pro*COBOL, Pro*FORTRAN, Pro*Pascal, SQL*Loader, SQL*Module, SQL*Net, SQL*Plus, Oracle7, Oracle Server, Oracle Server Manager, Oracle Call Interface, Oracle7 Enterprise Backup Utility, Oracle TRACE, Oracle WebServer, Oracle Web Application Server, Oracle Application Server, Oracle Network Manager, Secure Network Services, Oracle Parallel Server, Advanced Replication Option, Oracle Data Query, Cooperative Server Technology, Oracle Toolkit, Oracle MultiProtocol Interchange, Oracle Names, Oracle Book, Pro*C, and PL/SQL are trademarks or registered trademarks of Oracle Corporation. All other company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Contents

2 Designing Performant Applications

3 Sizing and Configuration

4 Tuning Oracle Application Server Components and Parameters

5 Monitoring Performance Statistics

A Operating System Tuning

Index

Preface

Audience

This book is for administrators who want to analyze and tune Oracle Application Server Release 4.0 for optimum performance.

The Oracle Application Server Documentation Set

This table lists the Oracle Application Server documentation set.

Title of Book	Part No.
Oracle Application Server 4.0.8 Documentation Set	A66971-03
Oracle Application Server Overview and Glossary	A60115-03
Oracle Application Server Installation Guide for Sun SPARC Solaris 2.x	A58755-03
Oracle Application Server Installation Guide for Windows NT	A58756-03
Oracle Application Server Administration Guide	A60172-03
Oracle Application Server Security Guide	A60116-03
Oracle Application Server Performance and Tuning Guide	A60120-03
Oracle Application Server Developer's Guide: PL/SQL and ODBC Applications	A66958-02
Oracle Application Server Developer's Guide: JServlet Applications	A73043-01
Oracle Application Server Developer's Guide: LiveHTML and Perl Applications	A66960-02
Oracle Application Server Developer's Guide: EJB, ECO/Java and CORBA Applications	A69966-01
Oracle Application Server Developer's Guide: C++ CORBA Applications	A70039-01
Oracle Application Server PL/SQL Web Toolkit Reference	A60123-03
Oracle Application Server PL/SQL Web Toolkit Quick Reference	A60119-03

Title of Book	Part No.
Oracle Application Server JServlet Toolkit Reference	A73045-01
Oracle Application Server JServlet Toolkit Quick Reference	A73044-01
Oracle Application Server Cartridge Management Framework	A58703-03
Oracle Application Server 4.0.8.1 Release Notes	A66106-04

Conventions

This table lists the typographical conventions used in this manual.

Convention	Example	Explanation
bold	**oas.h** **owsctl** **wrbcfg** **www.oracle.com**	Identifies file names, utilities, processes, and URLs
italics	*file1*	Identifies a variable in text; replace this place holder with a specific value or string.
angle brackets	`<filename>`	Identifies a variable in code; replace this place holder with a specific value or string.
courier	`owsctl start wrb`	Text to be entered exactly as it appears. Also used for functions.
square brackets	`[-c string]`	Identifies an optional item.
	`[on|off]`	Identifies a choice of optional items, each separated by a vertical bar (|), any one option can be specified.
braces	`{yes|no}`	Identifies a choice of mandatory items, each separated by a vertical bar (|).
ellipses	n,...	Indicates that the preceding item can be repeated any number of times.

The term “Oracle Server” refers to the database server product from Oracle Corporation.

The term **“oracle**” refers to an executable or account by that name.

The term “*oracle*” refers to the owner of the Oracle software.

Technical Support Information

Oracle Global Support can be reached at the following numbers:

- In the USA: **Telephone: 1.650.506.1500**
- In Europe: **Telephone: +44 1344 860160**
- In Asia-Pacific: **Telephone: +61. 3 9246 0400**

Please prepare the following information before you call, using this page as a check-list:

- ❏ your CSI number (if applicable) or full contact details, including any special project information
- ❏ the complete release numbers of the Oracle Application Server and associated products
- ❏ the operating system name and version number
- ❏ details of error codes and numbers and descriptions. Please write these down as they occur. They are critical in helping WWCS to quickly resolve your problem.
- ❏ a full description of the issue, including:
  - **What** - What happened? For example, the command used and its result.
  - **When** -When did it happen? For example, during peak system load, or after a certain command, or after an operating system upgrade.
  - **Where** -Where did it happen? For example, on a particular system or within a certain procedure or table.
  - **Extent** - What is the extent of the problem? For example, production system unavailable, or moderate impact but increasing with time, or minimal impact and stable.
- ❏ Keep copies of any trace files, core dumps, and redo log files recorded at or near the time of the incident. WWCS may need these to further investigate your problem. For a list of trace and log files, see “Configuration and Log Files” in the *Administration Guide.*

For installation-related problems, please have the following additional information available:

- ❏ listings of the contents of $ORACLE_HOME (Unix) or %ORACLE_HOME% (NT) and any staging area, if used.

- ❑ installation logs (**install.log**, **sql.log**, **make.log**, and **os.log**) typically stored in the **$ORACLE_HOME/orainst** (Unix) or **%ORACLE_HOME%\orainst** (NT) directory.

Documentation Sales and Client Relations

In the United States:

- To order hardcopy documentation, call Documentation Sales: **1.800.252.0303.**
- For shipping inquiries, product exchanges, or returns, call Client Relations: **1.650.506.1500.**

In the United Kingdom:

- To order hardcopy documentation, call Oracle Direct Response: **+44 990 332200.**
- For shipping inquiries and upgrade requests, call Customer Relations: **+44 990 622300.**

Reader's Comment Form

Oracle Application Server Performance and Tuning Guide

Part No. A60120-03

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

Oracle Application Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

If you would like a reply, please provide your name, address, and telephone number below:

Thank you for helping us improve our documentation.

1

Performance Overview

This chapter is intended to give you a brief overview of Oracle Application Server architecture and an introduction to tuning concepts.

Contents

Performance Terms

The following list describes performance terms used in this book:

Request latency	The time required to process a request.
Request throughput	The number of requests processed per unit of time.
Scalability	The ability to handle increasing numbers of requests without adversely affecting latency and throughput.

Oracle Application Server Overview

The Oracle Application Server is a sophisticated and highly tunable software product. Its flexibility allows you to make small adjustments that affect performance. By tuning your system, you can tailor its performance to best meet your needs. The intent of this chapter is to give you a general description of Oracle Application Server for performance and tuning purposes only. For a complete description of the product, see the *Oracle Application Server Overview*.

What is Performance Tuning?

Performance must be built in! Performance tuning cannot be performed optimally after a system is put into production. To achieve performance targets of response time, throughput, and constraints you must consider tuning for optimal performance during the phases of application analysis, design, and implementation. This section introduces some fundamental concepts:

- Response Time
- System Throughput
- Wait Time
- Critical Resources
- Effects of Excessive Demand
- Adjustments to Relieve Problems

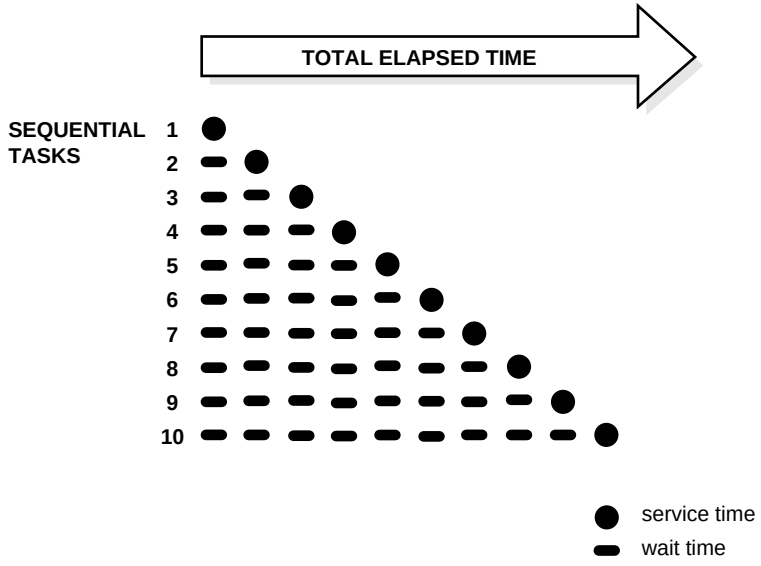
Response Time

Because response time equals service time plus wait time, you can increase performance in two ways:

- By reducing wait time
- By reducing service time

Figure 1–1 illustrates ten independent tasks competing for a single resource.

Figure 1–1 Sequential Processing of Multiple Independent Tasks



In this example only task 1 runs without having to wait. Task 2 must wait until task 1 has completed; task 3 must wait until tasks 1 and 2 have completed, and so on. (Although the figure shows the independent tasks as the same size, the size of the tasks will vary.)

Note: In parallel processing, if you have multiple resources, then more resources can be assigned to the tasks. Each independent task executes immediately using its own resource: no wait time is involved.

System Throughput

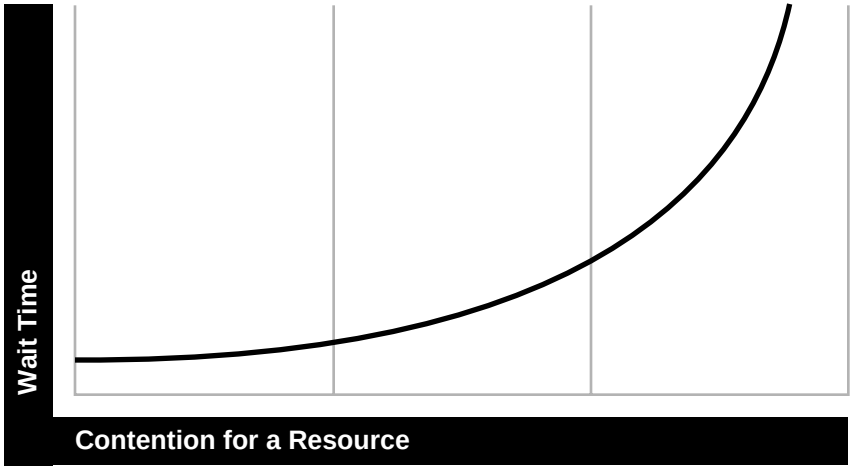
System throughput equals the amount of work accomplished in a given amount of time. Two techniques of increasing throughput exist:

- Get more work done with the same resources (reduce service time).
- Get the work done quicker by reducing overall response time. To do this, look at the wait time. You may be able to duplicate the resource for which all the users are waiting. For example, if the system is CPU bound you can add more CPUs. Keep in mind that distributing the load among multiple nodes should only be used when you are anticipating significant loads on the primary node.
- Schedule small tasks first

Wait Time

While the service time for a task may stay the same, wait time will go up as contention increases. If many users are waiting for a service that takes 1 second, the tenth user must wait 9 seconds for a service that takes 1 second.

Figure 1–2 Wait Time Rising with Increased Contention for a Resource



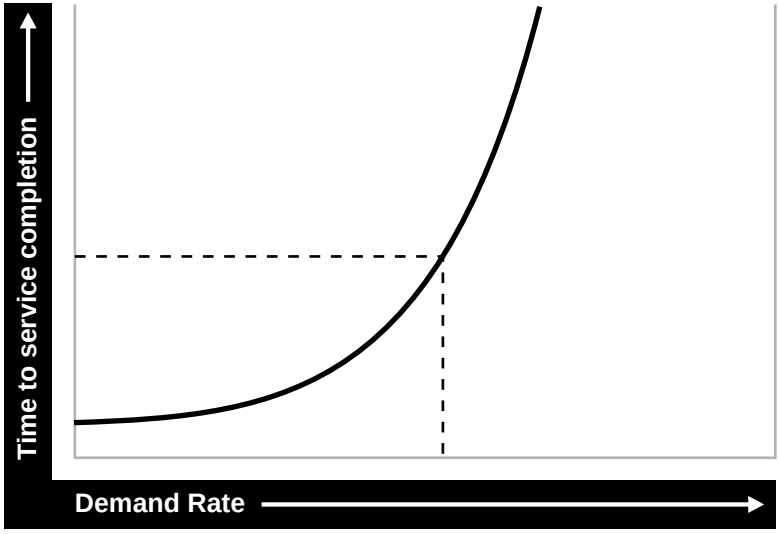
Critical Resources

Resources such as CPUs, memory, I/O capacity, and network bandwidth are key to reducing service time. Added resources increases throughput and reduces response time. Performance depends on the following:

- How many resources are available?
- How many clients need the resource?
- How long must they wait for the resource?
- How long do they hold the resource?

Figure 1–3 shows that as the number of units requested rises, the time to service completion rises.

Figure 1–3 Time to Service Completion vs. Demand Rate



To manage this situation, you have two options:

- You can limit demand rate to maintain acceptable response times.
- You can add resources, for example, another CPU or disk.

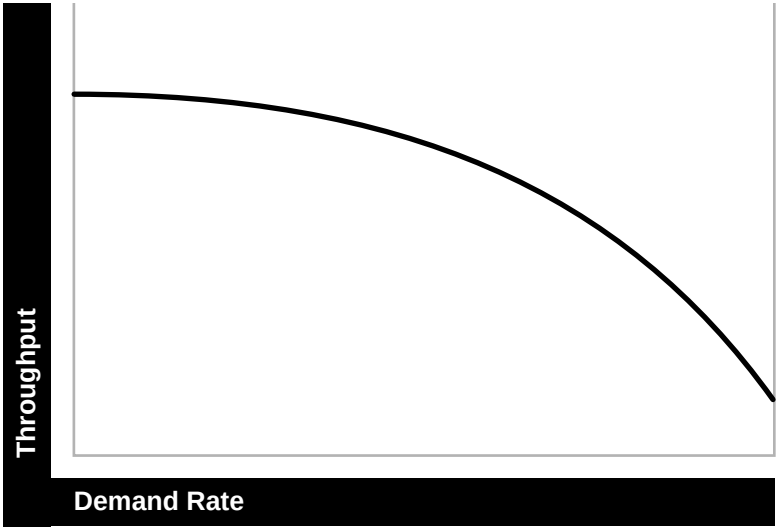
Effects of Excessive Demand

Excessive demand gives rise to:

- Greatly increased response time
- Reduced throughput

If there is any possibility of the demand rate exceeding the achievable throughput, a demand limiter is essential. Look at the possible demands that may be placed on the system and design the application or configure the system with these constraints in mind.

Figure 1–4 Increased Demand/Reduced Throughput



Adjustments to Relieve Problems

Performance problems can be relieved by making the following adjustments:

adjusting unit consumption	Some problems can be relieved by reducing resources per transaction or by reducing the service time. You can take other approaches, such as reducing the number of I/Os per transaction.
adjusting functional demand	Other problems can be abated by rescheduling or redistributing the work.
adjusting capacity	Problems may also be relieved by increasing or reallocating resources. If you start using multiple CPUs, going from a single CPU to a symmetric multiprocessor, you will have multiple resources available.

Setting Performance Targets

Whether you are designing or maintaining a system, you should set specific performance goals so that you know when to modify for optimal effectiveness. You can needlessly spend time tuning your system without significant gain if you attempt to alter parameters without a specific goal.

When designing your system, set a specific goal: for example, an order entry response time of less than three seconds. If the application does not meet that goal, identify the bottleneck causing the slowdown (for example, I/O contention), determine the cause, and take corrective action. During development, you should test the application to determine if it meets the designed performance goals before deploying the application.

Tuning usually involves a series of trade-offs. Once you have determined the bottlenecks, you may have to modify performance in some other areas to achieve the desired results. For example, if I/O is a problem, you may need to purchase more memory or more disks. If a purchase is not possible, you may have to limit the concurrency of the system to achieve the desired performance. However, if you have clearly defined goals for performance, the decision on what to trade for higher performance is simpler because you have identified the most important areas.

Setting User Expectations

Application developers, database administrators, and system administrators must be careful to set appropriate performance expectations for users. When the system carries out a particularly complicated operation, response time may be slower than when it is performing a simple operation. In cases like this, the slower response time is not unreasonable.

If an administrator should promise 1 second response time, consider how this might be interpreted. The administrator might mean that the operation would take 1 second in a web server--and might well be able to achieve this goal. However, users performing an operation or transaction over a network might experience a delay of a couple of seconds due to network traffic: they will not receive the response they expect in 1 second.

Evaluating Performance

With clearly defined performance goals, you can readily determine when performance tuning has been successful. Success depends on the functional objectives you have established with the user community, your ability to objectively measure whether or not the criteria are being met, and your ability to take corrective action to overcome any exceptions. The rest of this tuning manual describes the methodology and process of designing applications for performance tuning, with information about diagnostic tools and the types of corrective actions you can take.

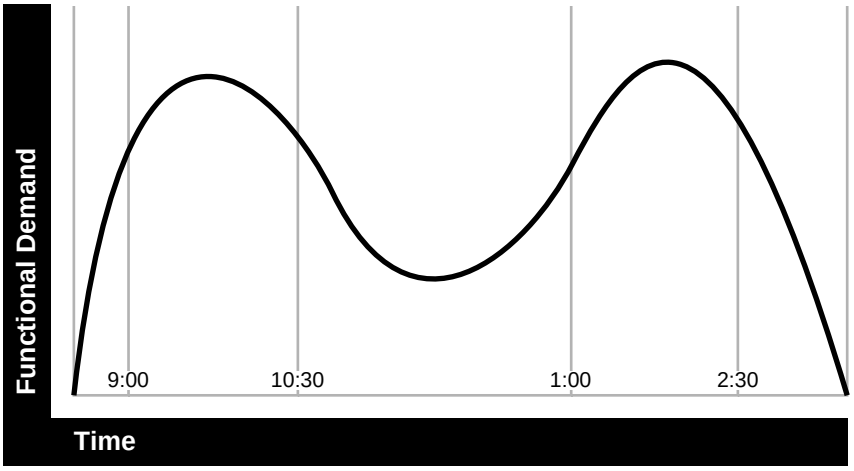
Administrators who are responsible for solving performance problems must keep a wide view of the all the factors that together determine response time. The perceived area of performance problems is frequently not the actual source of the problem. Users in the preceding example might conclude that there is a problem with Oracle Application Server, OAS, whereas the actual problem is with the network. Administrators must monitor the network, disk, CPU, and so on, to find the actual source of the problem--rather than simply assume that all performance problems stem from OAS.

Ongoing performance monitoring enables you to maintain a well tuned system. Keeping a history of the application's performance over time enables you to make useful comparisons. With data about actual resource consumption for a range of loads, you can conduct objective scalability studies and from these predict the resource requirements for load volumes you may anticipate in the future.

Performance Methodology

Achieving optimal effectiveness in your system requires planning, monitoring, and periodic adjusting. The first step in performance tuning is to determine the goals you need to achieve and to design effective usage of available technology into your applications with your goals in mind. After implementing your system, it is necessary to periodically monitor and adjust your system For example, you might want to ensure that 90% of the users should experience response times no greater than 5 seconds and the maximum response time any user should see is 20 seconds. Usually, it's not that simple. Your application may consist of a variety of operations, each with differing characteristics and acceptable response times. You will need to determine the acceptable response time for each potential operation and then estimate the likely mix of operations you expect from incoming users.

Figure 1–5 Adjusting Capacity and Functional Demand



You will also need to determine variances in the load at different times. For example, users might access the system heavily between 9:00am and 10:00am and then again between 1:00pm and 2:00pm. If your peak load occurs on a regular basis, for example, daily or weekly, the conventional wisdom is to configure and tune systems to meet your peak load requirements. The lucky users who access the application in off-time will typically achieve better response times than your peak-time users. If your peak load is infrequent, you may be willing to tolerate higher response times at peak loads for the cost savings of smaller hardware configurations.

Setting performance goals for new applications can be very difficult, because you're often guessing about the number of users who will be accessing the system and the operations those users will execute. If you're migrating an application from an older version of Oracle Application Server, it's slightly easier because you have an idea of what to expect. The number of users could increase unexpectedly, or users could start using new features of your application more heavily.

Regardless of whether you're starting from scratch or migrating an existing application, you will need to set some goals for performance, monitor your system over time, and adjust for growth or changes.

Roadmap to Improving Performance

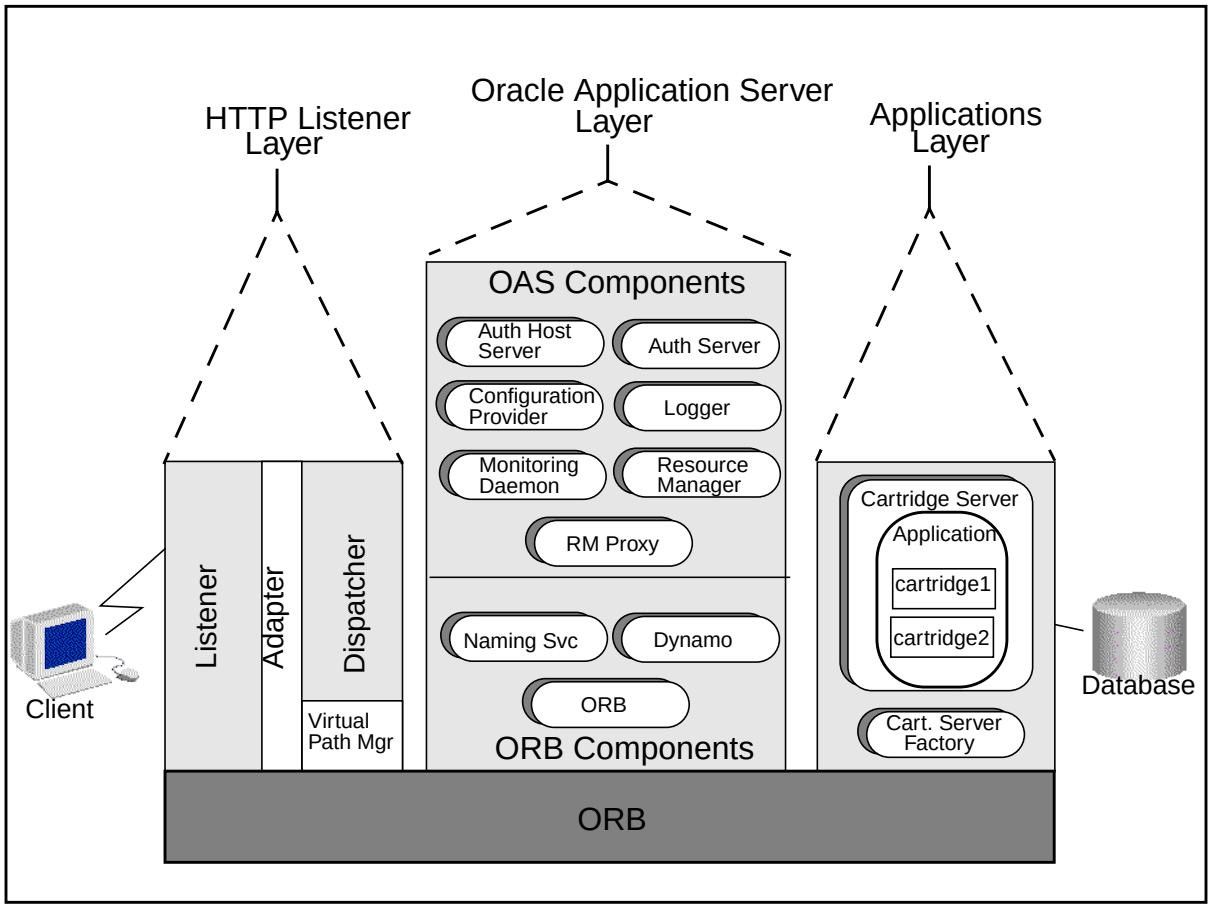
The process of performance tuning can be divided into a few basic areas:

- Application design: How to write applications that efficiently utilize hardware resources and handle increasing numbers of users effectively. For more information, see Chapter 2, “Designing Performant Applications”.
- Sizing and configuration: How much hardware do you need to support your performance goals. For more information, see Chapter 3, “Sizing and Configuration”.
- Parameter Tuning: How to set configurable parameters to achieve the best performance for your application. For more information, see Chapter 4, “Tuning Oracle Application Server Components and Parameters”.
- Performance Monitoring: How to determine what hardware resources are being used by your application and what response time your users are experiencing. For more information, see Chapter 5, “Monitoring Performance Statistics”.
- Troubleshooting: How to diagnose why an application is using more hardware resource than expected, or why the response time is greater than the desired goals.
- Since many Oracle Application Server applications include access to an Oracle database, we recommend consulting the Oracle RDBMS Tuning Guide (check reference) for guidelines on database performance tuning.

Architecture

Oracle Application Server can be used on a single, stand-alone machine or in a distributed environment among several machines. In a multiple node environment, the workload can be distributed, and performance can be greatly improved. Figure 1–6 shows the architecture of Oracle Application Server.

Figure 1–6 Oracle Application Server architecture



HTTP Listener Layer

The HTTP listener layer is made up of listeners, the adapter interface, and dispatchers.

Listeners

Listeners are HTTP servers; they handle incoming requests and route them to the dispatcher.

For a list of supported versions, refer to the product release notes.

Note that the documentation refers to the Oracle Application Server listener as the Web Listener and to all other supported listeners as third-party listeners. All of the above listeners are HTTP listeners.

For more information about listeners, see the *Oracle Application Server Administration Guide*.

Adapter

Oracle Application Server provides adapters to tightly integrate listeners with dispatchers. The adapter interface is a common API that both Oracle and third-party listeners use to connect to the dispatcher.

Dispatchers

When a listener receives an HTTP request that does not identify a static HTML page or CGI program, it passes the request to its dispatcher, which assigns the request to a cartridge instance of the appropriate type. This process is described in more detail in “Tracing Requests for Applications” on page 3-8. There is one dispatcher associated with each listener on each node of a Web site.

Virtual Path Manager

The dispatcher forwards requests to the virtual path manager. The virtual path manager maps a request to a cartridge type and passes this information back to the dispatcher. The virtual path manager also passes back authentication requirements to the dispatcher. The dispatcher can then contact the authentication server for authorization and forward the request on to the correct cartridge type.

Oracle Application Server Layer

The Oracle Application Server layer provides resource management in handling requests for applications deployed as cartridges on the server. It provides a common set of components for managing these applications. These components include load balancing, logging, automatic failure recovery, security, directory, and transaction components.

Applications Layer

The Applications layer is made up of applications, cartridges, and cartridge servers. For additional information, see "Introduction to Applications" in the *Oracle Application Server Administration Guide.*

Applications and Cartridges

Oracle Application Server applications consist of cartridges. Applications and cartridges are the two main objects that you use when building applications for the application server environment.

A cartridge consists of code that executes application logic and configuration data that enable it to locate the application logic. For example, the PL/SQL cartridge contains code that enables it to connect to Oracle databases and execute PL/SQL stored procedures in the database. The configuration data in the cartridge contains information such as which Oracle database to connect to and what username/password to use in order to run that stored procedure. Cartridges also enable your application to communicate with other components of the application server.

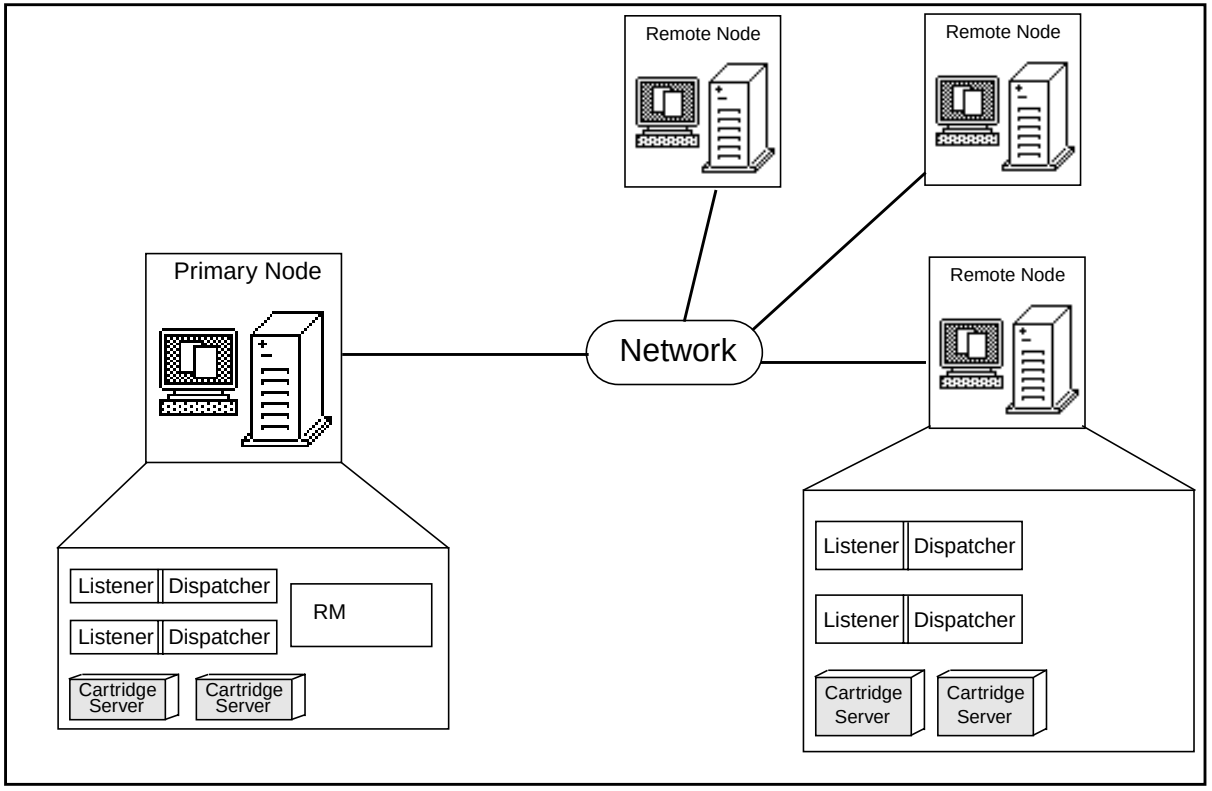
Cartridges can provide runtime environments for specific programming languages. For example, the JServlet cartridge contains a Java Virtual Machine for running Java class files, and the Perl cartridge contains a Perl interpreter for running Perl scripts. See “Creating Applications” on page 3-1 for a list of cartridges.

Cartridges are contained within applications. An application contains one or more cartridges, and within an application, all the cartridges must be derived from the same cartridge type. You cannot have, for example, an application that contains both PL/SQL and JServlet cartridges.

Multi-node Architecture

Figure 1–7 illustrates the architecture of a multi-node Oracle Application Server site. A multi-node site is a collection of Oracle Application Server machines that together form a distributed platform for server-side applications. For each site, a single machine, referred to as the "primary node", hosts the Resource Manager (RM) proxy, which stores configuration data for the entire site. The other machines that comprise the site are called "remote nodes".

Figure 1–7 Oracle Application Server multi-node site



Depending on the hardware configuration of each node, and the workload your site supports, you can configure different nodes to run different Oracle Application Server components.

Distributing Load Among Multiple Nodes

For large workloads, you can improve performance by setting up your site with multiple nodes, running listeners and dispatchers on the primary node, and running cartridge servers on remote nodes. You can optimize performance by dividing responsibilities among nodes according to the workloads you anticipate.

For example, if your site offers a large volume of data for File Transfer Protocol (FTP) download, you will get better performance for these downloads, and other server accesses, by hosting the downloadable data on a separate Oracle Application Server machine. This frees resources on other machines for handling HTTP and application requests.

Distributed versus Single-Node Configurations

Oracle Application Server provides support for load balancing requests across multiple nodes in a site. However, if you are not sufficiently using the hardware resources on one node, it is unlikely that you will see an increase in performance from adding a second node. You may still wish to deploy a distributed configuration for reasons other than performance, such as failure recovery or security. For performance, add an additional node to an existing site only when you have saturated your existing hardware resources.

New Performance Features in Oracle Application Server 4.0.8

Oracle Application Server 4.0.8 contains several new features that improve performance and scalability.

Policy Management: In earlier versions of Oracle Application Server, configurable parameters controlled the number of minimum and maximum server processes, cartridge instances and execution threads. In 4.0.8., Oracle Application Server also includes a priority mode for load balancing. In priority mode (the default), the application server will manage creating new processes, threads and instances. Tuning minimums and maximums can be a complicated activity, and many performance problems in earlier versions of Oracle Application Server were caused by incorrect use of these parameters. Priority mode simplifies administration and provides more dynamic load balancing. It is the recommended setting.

Dynamic Monitoring: Oracle Application Server 4.0.8 includes a new component, the Dynamic Monitoring Service, DMS, which extends the capabilities of the administrative monitoring page in previous releases. DMS provides a wider range of performance metrics and a richer user interface. The administrative monitoring page is still supported in addition to DMS.

Session Management for Java Servlets: User session management for Java Servlet (also new in Oracle Application Server 4.0.8) applications has improved performance based on a new session cache facility. The Java Servlet Developer's Guide provides details on this new feature.

2

Designing Performant Applications

In designing applications, performance is only one consideration. There are often trade-offs made to maintainability, security, and ease of programming when maximizing performance. This chapter provides guidelines for writing applications that balance performance with these other considerations.

Contents

Java-Based Applications

Oracle Application Server offers a variety of Java programming models. These are described in *Oracle Application Server: Overview and Glossary*. This section will focus on database connectivity and the two most commonly used models.

Database Connectivity

With Oracle Application Server there are two JDBC drivers for database access, the OCI and the JTS driver. For better performance, use the OCI driver when your application:

- uses programmatic instead of declarative transactions. Programmatic transactions require that you include `COMMIT` statements explicitly in your code

instead of specifying transaction properties on the Node Manager configuration forms.

- does not require transactions that span multiple method invocations
- does not require distributed transactions that access multiple databases within one transaction.

There is a substantial performance gain in using the OCI driver in these situations.

Using programmatic transactions instead of declarative transactions often results in better performance because declarative transactions are slower and require the JTS driver. See "Enabling Transactions" in the *Administration Guide* for further details about using transactions with Oracle Application Server.

JServlet Applications

JServlets are the simplest Java programming model offered with Oracle Application Server.

Threading Models

Developer's Guide: JServlet Applications provides information on threading models supported by the JServlet cartridge. The following threading issues should be noted.

- Implementing the `SingleThreadModel` interface does not imply that only one thread will be used per process. It only determines whether concurrent users share access to servlet instances or whether each executing thread has its own instance.
- Implementing the `SingleThreadModel` interface creates servlets that are serially reusable. Although an individual servlet instance can only execute one request at a time, when the request completes the servlet will be available to handle another request.
- Declaring objects as static will cause the objects to be shared by all threads in the process regardless of threading model.
- Spawning sub-threads is not recommended because it limits Oracle Application Server's load balancing and management capabilities.

For most applications, servlets that implement the SingleThreadModel interface are recommended, particularly for applications that access an Oracle database. These applications are easier to develop and can achieve the same level of performance as a multi-threaded servlet.

Example 2–1 shows a servlet that implements the `SingleThreadModel` interface and connects to a database. It demonstrates the following performance concepts:

1. Implementing the `SingleThreadModel` allows each thread to have its own servlet instance during execution. Since each instance will connect to a database in the `init()` method and closes the connection in the `destroy()` method, the database connection is kept for the life cycle of the servlet instance. The cost of opening and closing the database connection is only incurred once per process. Each instance within a process will get its own database connection and all instances within a process can simultaneously communicate with the database.

   If the `SingleThreadModel` is implemented, access to database connections would need to be synchronized because the connections are shared by all instances in the process. You can create multiple connections yourself, but your application needs to keep track of and synchronize access to the connections.

2. If the database connection is lost while a servlet instance is still alive, attempting to use the connection will raise the `SQLException` exception. If the connection is not re-established, all subsequent requests sent to this servlet instance will fail. Additional error checking could be added to check for connection failures and call `getDBConnection()` to establish a new connection. This will make the application more robust.

3. The prepare statement used to access the database is created in the `init()` method through the `getDBConnection()` method. The constant portion of the prepare statement is set up once, only the `setString` and `execute` statements are in the `doGet()` method because they must be declared and called once for each request.

   Similar to the database connection, the prepare statement is not closed after each request and thus, can be reused. It is closed in the `destroy()` method.

Example 2–1 Implementing the SingleThreadModel interface

```
import java.io.*;
import java.lang.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class JSampleSingleThread extends HttpServlet
     implements SingleThreadModel { // NOTE 1 - implement the SingleThreadModel
  private final static String COMPONENT_NAME = "JSampleSingleThread";
  private final static String DBSERVER_PARAM = "DBServer";
  private final static String USERID         = "userid";

  // Instance variables
  private String dbServer = null;
  private Connection dbConn = null;
  private CallableStatement dbStmt = null;

  //---------------------------------------------------------------------------
  // Servlet life cycle methods

  // The init() method is called every time when an instance of this
  // servlet is created.
  public void init(ServletConfig config) throws ServletException {
    super.init(config);

    // Get configuration information
    dbServer = config.getInitParameter(DBSERVER_PARAM);
    if (dbServer == null) {
      throw new ServletException("getInitParameter Exception: " +
         DBSERVER_PARAM + " " + dbServer);
    }

    // get the database connection
    try {
      getDBConnection();
    } catch (Throwable e) {
      throw new ServletException(e.toString());
    }
  } // init
```

```
// The destroy() method is called every time when an instance of this
// servlet is destroyed.
public void destroy()
{
  // Close the database statement and connection used by this instance.
  try {
    if (dbStmt != null)
      dbStmt.close();

    if (dbConn != null)
      dbConn.close();
  } catch (SQLException e) {
    getServletContext().log(e.toString());
  }
} // destroy

//---------------------------------------------------------------------------
// doGet() is the servlet's main entry for every request.
public void doGet (HttpServletRequest httpRequest,
                   HttpServletResponse httpResponse)
            throws ServletException, IOException {
  String userName = null;
  String employer = null;
  String allowances = null;
  String addWithholding = null;
  PrintWriter httpOut;

  // Initialize HTTP response header
  httpResponse.setContentType("text/html");
  httpOut = httpResponse.getWriter();
  httpOut.println("<html>");
  httpOut.println("<head><title>" + COMPONENT_NAME + "</title></head>");

  // Get user id parameter
  String userid = httpRequest.getParameter(USERID);
  if (userid == null) {
    pgError(httpOut, "Unable to retrieve user information, " +
      USERID + " is null");
    return;
  }
```

```
  // Get data from database, most of dbStmt is defined in getDBConnection()
  try {
    dbStmt.setString (1, userid);
    dbStmt.execute();
    String retCode = dbStmt.getString(8).trim();

    if (retCode.equals("0")) {
      String lastName   = (dbStmt.getString(2)).trim();
      String firstName  = (dbStmt.getString(3)).trim();
      String middleName = (dbStmt.getString(4)).trim();
      employer          = (dbStmt.getString(5)).trim();
      allowances        = (dbStmt.getString(6)).trim();
      addWithholding    = (dbStmt.getString(7)).trim();
      userName = firstName + " " + middleName + " " + lastName;
      pgOut(httpOut, userid, userName, employer, allowances, addWithholding);
    } else if (retCode.equals("100")) {
      pgError(httpOut, "Record not found for " + userid);
    } else {
      pgError(httpOut, "Unable to retrieve record for " + userid +
                       " (SQLERROR " + retCode);
    }
  } catch (SQLException se) {
    // NOTE 2 - catch the SQLException exception
    throw new ServletException(se.toString());
  } catch (Throwable e) {
    throw new ServletException(e.toString());
  } // try
} // doGet


//--------------------------------------------------------------------------
// Servlet private methods

// Get database connection and prepare the database statement
// This method is called from the init() method.
private void getDBConnection()
throws Throwable {
  try { // open the database connection
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    dbConn = DriverManager.getConnection(
               "jdbc:oracle:oci8:@" + dbServer, "scott", "tiger");
  } catch (Throwable e) {
    throw e;
  }
```

```
    try { // NOTE 3 - set up prepare statement
      dbStmt =
        dbConn.prepareCall("begin GET_PERFSAMPLE_DATA(?,?,?,?,?,?,?,?); end;");
      dbStmt.registerOutParameter (2, Types.VARCHAR);
      dbStmt.registerOutParameter (3, Types.VARCHAR);
      dbStmt.registerOutParameter (4, Types.VARCHAR);
      dbStmt.registerOutParameter (5, Types.VARCHAR);
      dbStmt.registerOutParameter (6, Types.VARCHAR);
      dbStmt.registerOutParameter (7, Types.VARCHAR);
      dbStmt.registerOutParameter (8, Types.VARCHAR);
    } catch (Throwable e) {
      throw e;
    }
  } //getDBConnection


  // Return the HTML page
  private void pgOut (PrintWriter httpOut, String userid,
    String userName, String employer, String allowances,
    String addWithholding) {
    httpOut.println("<body>");
    httpOut.println("<h1>User: <font color=blue>" +
                    userid + "</font> </h1>");
    httpOut.println("<h1>Name: <font color=blue>" +
                    userName + "</font> </h1>");
    httpOut.println("<h1>Employer: <font color=blue>" +
                    employer + "</font> </h1>");
    httpOut.println("<h1>Number of Allowances: <font color=blue>" +
                    allowances + "</font> </h1>");
    httpOut.println("<h1>Additional Amount of Withholding: <font color=blue>" +
                    addWithholding + "</font> </h1>");
    httpOut.println("</body>");
  } // pgOut


  // Print HTML error page
  private void pgError (PrintWriter httpOut, String errmsg) {
    httpOut.println("<body>");
    httpOut.println("<h1> <font color=red> ERROR: " + errmsg +
                   "</font> </h1>");
    httpOut.println("<hr>");
    httpOut.println("</body>");
  } // pgError
} // JSampleSingleThread
```

Example Notes

This example reads a user id from the invoking URL and retrieves information associated with that id from a database. The application is invoked with the URL:

http://<*machine*>:<*port*>/<*virtual_path*>/JSampleSingleThread?userid=<*id*>

The following values in the code will need to be modified to match your system's configuration.

- *DBServer* (used in the static declarations) is a value specified in the Java Environment form for the application. It should be defined as the following name-value pair:

  name — Servlet.JSampleSingleThread.initArgs

  value — DBServer=<*tnsnames.ora service name*>

- *scott* and *tiger* are used in the `getDBConnection()` method. These are parameters passed in the connect string that represent the user id and password for accessing the database. Change these to values appropriate for your system.

The information in the database is stored according to the schema shown in Example 2–2.

Example 2–2 Schema for perfSample

```
create table perfSample
( user_id                varchar2(30),
  last_name              varchar2(30),
  first_name             varchar2(30),
  middle_name            varchar2(30),
  employer               varchar2(30)  not null,
  allowances             number(2)     not null,
  addWithholding         number(10,2)  not null,
  constraint             userid_pkey primary key(user_id)
);
```

Sessions

When an incoming request is received for a user with an open session, Oracle Application Server will select an instance to handle the request and then load the necessary session data into that instance. If the session is initialized as a local session, each session is restricted to a single servlet process. However, any instance within that process can execute requests for that session. For better performance, use local sessions.

Example 2–3 shows a simple JServlet application which implements the SingleThreadModel and uses the session manager to store intermediate session data before updating them to the database.

The example uses the following model:

1. Get a user id from a client. Retrieve the appropriate data from the database and save it in the session manager.
2. Let the client change the data and request that the change be updated in the database. Save the changed data in the session manager.
3. Ask the client to confirm the changes.
4. When the client requests that the changed data be committed to the database, retrieve the data from the session manager and commit it to the database.
5. Invalidate the session.

The following items are configurable items that can have an effect on the performance of your session based application.

- **Memory usage** — Using sessions consumes memory. The amount of memory per process will depend on the session timeout value, the size and number of objects that can be stored in the session manager, and the maximum number of clients per servlet. Applications that use sessions should be configured with sufficient memory. To increase the heap size in the servlet process, set the `INITIAL_HEAP` and `MAX_HEAP` variables to appropriate values when configuring the servlet. These variables can be set in the JServlet application's Java Environment form. See the *Administration Guide* for more information.
- **Session invalidation** — A session can only be invalidated by calling the `session.invalidate()` or if it times out. If the number of available sessions runs out and new servers cannot be started (the maximum number of servers is reached), new clients would have to wait until sessions have been released or invalidated. Be sure to allow for a sufficient number of *concurrent sessions* in the JServlet cartridge's Tuning form. See the *Administration Guide* for more information.

  Concurrent sessions are defined by the maximum number of clients per server times the maximum number of servers for the servlet.

Example 2–3 Using sessions

```
import java.io.*;
import java.lang.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class JSampleSession extends HttpServlet implements SingleThreadModel {
  // String constants
  private final static String COMPONENT_NAME = "JSampleSession";
  private final static String DBSERVER_PARAM = "DBServer";
  private final static String USERID         = "userid";
  private final static String REQID          = "reqid";
  private final static String EMPLOYER       = "employer";
  private final static String ALLOWANCES     = "allowances";
  private final static String AMOUNT         = "amount";
  private final static String REQID_GET      = "Get DB Data";
  private final static String REQID_UPDATE   = "Submit Changes";
  private final static String REQID_COMMIT   = "Commit Changes";
  private final static String REQID_RESET    = "Reset to DB values";

  // Instance variables
  private String dbServer = null;
  private Connection dbConn = null;
  private CallableStatement dbGetStmt = null;
  private CallableStatement dbPutStmt = null;


  // The init() function is called every time when an instance of this servlet
  // is created.
  public void init(ServletConfig config) throws ServletException {
    super.init(config);

    // Get configuration information
    dbServer = config.getInitParameter(DBSERVER_PARAM);
    if (dbServer == null) {
      throw new ServletException("getInitParameter Exception: " +
        DBSERVER_PARAM + "=" + dbServer);
    }
```

```
  // Get database connection
  try {
    getDBConnection();
  } catch (Throwable e) {
    throw new ServletException(e.toString());
  }
} // init()


// The destroy() method is called every time when an instance of this
// servlet is destroyed.
public void destroy() {
  // Close the database connection used by this instance.
  try {
    if (dbGetStmt != null) dbGetStmt.close();
    if (dbPutStmt != null) dbPutStmt.close();
    if (dbConn != null)    dbConn.close();
  } catch (SQLException e) {
    getServletContext().log(e.toString());
  } // try
} // destroy()


// The doGet() method is this servlet's main entry point for every request
public void doGet (HttpServletRequest httpRequest,
                   HttpServletResponse httpResponse)
       throws ServletException, IOException {
  PerfSampleData dbData  = null;
  PerfSampleData newData = null;
  boolean getDBData = false;
  HttpSession session;
  PrintWriter httpOut;

  // Initialize HTTP response header
  httpResponse.setContentType("text/html");
  httpOut = httpResponse.getWriter();
  httpOut.println("<html>");
  httpOut.println("<head><title>" + COMPONENT_NAME + "</title></head>");

  // Get input parameters
  String userid = httpRequest.getParameter(USERID);
  if (userid == null) {
    pgError(httpOut, USERID + " is null");
    return;
  }
```

```
String reqid = httpRequest.getParameter(REQID);
if (reqid == null) {
  reqid = REQID_GET;
}

// Get the servlet session.
// This function returns a new session or an existing session
// established by the client in previous requests.
try {
  session = httpRequest.getSession();
} catch (Exception e) {
  throw new ServletException(e.toString());
}

// Create session keys
String useridKey  = "USERID";
String dbDataKey  = userid + "DBDATA";
String newDataKey = userid + "NEWDATA";

// If this is not a new session, check if the userid is the
// same as last userid.  If not, invalidate the old session
// and set up a new one for the new user.
if (!session.isNew() &&
    !userid.equals((String)session.getValue(useridKey))) {
  try {
    session.invalidate();
    session = httpRequest.getSession();
  } catch (Exception e) {
    throw new ServletException(e.toString());
  } // try
  getDBData = true;
} // if

// If this is a new session, retrieve the data from the
// database and save it in the session manager.
// Otherwise, retrieve the data from session manager.
if (session.isNew() || getDBData) {
  try {
    dbData = getPerfSampleData(userid, httpOut);
    session.putValue(dbDataKey, (Object)dbData);
    session.putValue(useridKey, (Object)userid);
```

```
  } catch (Throwable e) {
    session.invalidate();
    throw new ServletException(e.toString());
  } // try
} else { // the session is not new
  try {
    dbData = (PerfSampleData) session.getValue(dbDataKey);
  } catch (Exception e) {
    throw new ServletException(e.toString());
  } // try
} // if-else

// The following if block looks at the requested functions and
// follows the appropriate branch based on the value of reqid.

// Get data from database
if (reqid.equals(REQID_GET)) {
  returnPage(httpOut, httpRequest, dbData, REQID_GET);
}

// Save data to session manager and ask user to confirm
else if (reqid.equals(REQID_UPDATE)) {
  newData = new PerfSampleData();
  newData.userid = dbData.userid;
  newData.lastName = dbData.lastName;
  newData.firstName = dbData.firstName;
  newData.middleName = dbData.middleName;
  newData.employer = httpRequest.getParameter(EMPLOYER);
  newData.allowances = httpRequest.getParameter(ALLOWANCES);
  newData.withholding = httpRequest.getParameter(AMOUNT);

  if (newData.employer.length() == 0  ||
      newData.allowances.length() == 0 ||
      newData.withholding.length() == 0) {
    pgError(httpOut, "One or more data fields are null");
    returnPage(httpOut, httpRequest, newData, REQID_GET);
    return;
  } // if
```

```
  try {
    Integer.parseInt(newData.allowances);
  } catch (NumberFormatException e) {
    pgError(httpOut, "Number of nxemptions value is invalid");
    returnPage(httpOut, httpRequest, newData, REQID_GET);
    return;
  } // try

  try {
    Float.valueOf(newData.withholding);
  } catch (NumberFormatException e) {
    pgError(httpOut, "Additional withholding amount is invalid");
    returnPage(httpOut, httpRequest, newData, REQID_GET);
    return;
  } // try

  try {
    session.putValue(newDataKey, (Object)newData);
  } catch (Exception e) {
    throw new ServletException(e.toString());
  } // try
  returnPage(httpOut, httpRequest, newData, REQID_UPDATE);
} // else if reqid.equals(REQID_UPDATE)

// Commit the changes to the database and invalidate the session.
else if (reqid.equals(REQID_COMMIT)) {
  try {
    newData = (PerfSampleData) session.getValue(newDataKey);
    if (newData == null) {
      throw new ServletException(
         "getValue Exception: " + newDataKey + " not found in session");
    }

    String retCode = updatePerfSampleData(newData, httpOut);
    if (retCode.equals("0")) {
      returnPage(httpOut, httpRequest, newData, REQID_COMMIT);
      session.invalidate();
    } else {
      returnPage(httpOut, httpRequest, newData, REQID_GET);
    } // if-else
  } catch (Throwable e) {
    throw new ServletException(e.toString());
  } // try
} // else if reqid.equals(REQID_COMMIT)
```

```
  // Reset to database value
  else if (reqid.equals(REQID_RESET)) {
    returnPage(httpOut, httpRequest, dbData, REQID_RESET);
  } // else if reqid.equals(REQID_RESET)

  // Unknown request
  else {
    pgError(httpOut, "Invalid request");
  } // if reqid.equals() block
} // doGet()


// Get data from database
private PerfSampleData getPerfSampleData(String userid, PrintWriter httpOut)
throws Throwable {
  PerfSampleData dbData = new PerfSampleData();

  try {
    dbGetStmt.setString (1, userid);
    dbGetStmt.execute();

    String retCode = dbGetStmt.getString(8).trim();

    if (retCode.equals("0")) {
      dbData.userid      = userid;
      dbData.lastName    = (dbGetStmt.getString(2)).trim();
      dbData.firstName   = (dbGetStmt.getString(3)).trim();
      dbData.middleName  = (dbGetStmt.getString(4)).trim();
      dbData.employer    = (dbGetStmt.getString(5)).trim();
      dbData.allowances  = (dbGetStmt.getString(6)).trim();
      dbData.withholding = (dbGetStmt.getString(7)).trim();
    } else {
      if (retCode.equals("100")) {
        pgError(httpOut, "Record not found for " + userid);
      }  else {
        pgError(httpOut, "Unable to retrieve record for " + userid +
                         " (SQLERROR " + retCode + ")");
      } // inner if-else
    } // outer if-else
  } catch (SQLException e) {
    throw e;
  } // try

  return dbData;
} // getPerfSampleData()
```

```
// Update data to database
private String updatePerfSampleData( PerfSampleData dbData,
                                     PrintWriter httpOut) throws Throwable {
  String retCode = null;

  try {
    dbPutStmt.setString (1, dbData.userid);
    dbPutStmt.setString (2, dbData.employer);
    dbPutStmt.setString (3, dbData.allowances);
    dbPutStmt.setString (4, dbData.withholding);
    dbPutStmt.execute();

    retCode = dbPutStmt.getString(5).trim();
    if (!retCode.equals("0")) {
      pgError(httpOut, "Unable to update to database, SQL returns error " +
              retCode);
    } // if
  } catch (SQLException e) {
    throw e;
  } // try
  return retCode;
} // updatePerfSampleData()


// Get database connection
private void getDBConnection() throws Throwable {
  try {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    dbConn = DriverManager.getConnection(
               "jdbc:oracle:oci8:@" + dbServer, "scott", "tiger");

    dbGetStmt =
      dbConn.prepareCall("begin GET_PERFSAMPLE_DATA(?,?,?,?,?,?,?,?); end;");
    dbGetStmt.registerOutParameter (2, Types.VARCHAR);
    dbGetStmt.registerOutParameter (3, Types.VARCHAR);
    dbGetStmt.registerOutParameter (4, Types.VARCHAR);
    dbGetStmt.registerOutParameter (5, Types.VARCHAR);
    dbGetStmt.registerOutParameter (6, Types.VARCHAR);
    dbGetStmt.registerOutParameter (7, Types.VARCHAR);
    dbGetStmt.registerOutParameter (8, Types.VARCHAR);
```

```
    dbPutStmt =
      dbConn.prepareCall("begin UPDATE_PERFSAMPLE_DATA (?,?,?,?,?); end;");
    dbPutStmt.registerOutParameter (5, Types.VARCHAR);
  } catch (Throwable e) {
    throw e;
  } // try
} // getDBConnection()


// Return the HTML page.
public void returnPage( PrintWriter httpOut,   HttpServletRequest httpRequest,
                        PerfSampleData dbData, String reqid) {
  httpOut.println("<body>");

  String userName = dbData.firstName + " " + dbData.middleName + " " +
                    dbData.lastName;

  httpOut.println("<h1>User: " + dbData.userid + "</h1>");
  httpOut.println("<h1>Name: " + userName + "</h1>");

  httpOut.println("<form method=get action=" +
                  HttpUtils.getRequestURL(httpRequest).toString() + ">");

  // show this if you want to confirm
  if (reqid.equals(REQID_UPDATE)) {
    httpOut.println("<h1> <font size=12 color=purple> " +
                    "Please confirm your changes: " +
                    "</font> </h1>");
    httpOut.println("<h1>Employer: <font color=blue>" +
                    dbData.employer + "</font> </h1>");
    httpOut.println("<h1>Number of Allowances: <font color=blue>" +
                    dbData.allowances +
                    "</font> </h1>");
    httpOut.println("<h1>Additional Amount of Withholding: $<font color=blue>"
                    + dbData.withholding +
                    "</font> </h1>");
    httpOut.println("<hr>");
    httpOut.println("<input type=submit name=" + REQID +
                    " value='" + REQID_COMMIT + "'>");
    httpOut.println("<input type=submit name=" + REQID +
                    " value='" + REQID_RESET + "'>");

    // show this after submitting changes
  } else if (reqid.equals(REQID_COMMIT)) {
    httpOut.println("<h1> <font size=12 color=purple> " +
```

```
                    "Changes updated to database." +
                    "</font> </h1>");
      httpOut.println("<input type=submit name=" + REQID +
                    " value='" + REQID_GET + "'>");

    // show this to ask for changes
    } else {
      httpOut.println("<h1>Employer <input type=text size=30 name=" +
                    EMPLOYER + " value=" + dbData.employer + "> </h1>" );
      httpOut.println("<h1>Number of Allowances <input type=text size=2 name="
                    + ALLOWANCES + " value=" + dbData.allowances + "> </h1>");
      httpOut.println("<h1>Additional Amount of Withholding" +
                    "<input type=text size=15 name=" + AMOUNT +
                    " value=" + dbData.withholding + "> </h1>");
      httpOut.println("<hr>");

      httpOut.println("<input type=submit name=" + REQID +
                    " value='" + REQID_UPDATE + "'>");
      httpOut.println("<input type=submit name=" + REQID +
                    " value='" + REQID_RESET + "'>");
    } // if-else if-else

    httpOut.println("<input type=hidden name=" + USERID +
                   " value=" + dbData.userid +">");
    httpOut.println("</form>");
    httpOut.println("</body>");
  } // returnPage()


  // Print HTML error page
  private void pgError (PrintWriter httpOut, String errmsg) {
    httpOut.println("<body>");
    httpOut.println("<h1> <font color=red> ERROR: " + errmsg +
                   "</font> </h1>");
    httpOut.println("<hr>");
    httpOut.println("</body>");
  } // pgError()
} JSampleSession class
```

```
// Data object to be stored in session manager
class PerfSampleData {
  public String userid;
  public String firstName;
  public String middleName;
  public String lastName;
  public String employer;
  public String allowances;
  public String withholding;
} // PerfSampleData class
```

See "Example Notes" on page 2-8 for information about invoking and configuring this example.

Enterprise Java Beans invoked by JServlet Applications

For most HTTP clients, implementing applications completely in JServlets instead of invoking Enterprise Java Beans (EJBs) from JServlet applications will improve performance. However, there are scenarios where it is appropriate or necessary to invoke EJBs directly.

Stateless vs. Stateful Beans

A stateful bean exists for a single client. It carries a "state" that is relevant only for the particular client. A bean is instantiated for every remote interface instance held by the client. Only one thread will execute at a time for each session bean, but multiple beans can execute concurrently in the same server process.

A stateless bean does not carry any state and thus, can service multiple clients — one at a time per bean. The number of beans instantiated depends on the number of concurrent method invocations.

Oracle recommends the use of stateless beans wherever possible. These beans are pooled and after being used, beans are returned to the pool for reuse. This improves performance because the number of beans that need to be maintained is based on the request rate and not on the number of users in the system. By pooling beans, the cost of creating and removing beans is incurred only once per bean.

Example 2–4 shows an EJB that can be invoked from a JServlet application (shown in Example 2–6). In this example, accessing the database now occurs in the EJB instead of the JServlet as opposed to Example 2–1. Each bean instance will open one database connection in the `ejbCreate()` method and close it in the `ejbRemove()` method when the bean instance is destroyed.

For optimal performance, this bean should be declared as stateless in the deployment descriptor file. This gives each bean instance its own database connection and allows it to service multiple client requests. Although each bean must open its own database connection, the beans will be pooled after servicing their requests and will not need to reopen the database connection for future requests.

The information in the database is stored according to the schema shown in Example 2–2.

Example 2–4 Stateless EJB application

> **Note:** Only the main EJB application is shown here. The other supporting files (home, remote, exception) are in the samples directory on the Oracle Application Server CD. The samples path can be found in the Release Notes.

```
package ejbSample;

import oracle.oas.ejb.*;
import java.util.*;
import java.io.*;
import java.sql.*;
import javax.ejb.*;
import javax.naming.*;

public class ejbSampleDB implements javax.ejb.SessionBean {
  // Constants
  private static final String DBSERVER = "DBServer";
  private static final String OAS_LOGGER = "oas_service:logger";

  // Instance variables
  Logger logger = null;
  SessionContext sessctx = null;
  String dbServer = null;
  Connection dbConn = null;
  CallableStatement dbStmt = null;

  //  EJB constructor and life cycle methods
  public void setSessionContext(SessionContext sc) {
    this.sessctx = sc;
  }
```

```
public void ejbCreate() throws CreateException {

  try {
    logger = (Logger) (new InitialContext()).lookup(OAS_LOGGER);
    logger.setSeverity(Logger.LOG_SEVERITY_ERROR);
  } catch(javax.naming.NamingException e) {
    e.printStackTrace (System.out);
    throw new CreateException (e.getMessage());
  }

  Properties env = sessctx.getEnvironment();
  dbServer = env.getProperty(DBSERVER);

  if (dbServer == null) {
    logger.println(DBSERVER + " is null");
    throw new CreateException();
  }

  try {
    getDBConnection();  // Get db connection and set up the prepare statement
  } catch (Throwable e) {
    throw new CreateException();
  } // try
} // ejbCreate


public void ejbRemove() {
  // Close the DB connection before exit
  try {
    if (dbStmt != null) { dbStmt.close(); }
    if (dbConn != null) { dbConn.close(); }
  } catch (Throwable e) {
    logger.println(e.toString());
  } // try
} // ejbRemove
```

Note: The `ejbActivate()` and `ejbPassivate()` methods are not called in this release. In future releases when these methods are called, the database connection must be closed in the `ejbPassivate()` method and reopened in the `ejbActivate()` method because database connections are not serializable with stateless beans.

```
public void ejbActivate() { }
public void ejbPassivate() { }


//---------------------------------------------------------------------
//  get data from database
public ejbSampleDBObj getSampleData(String userid)
     throws ejbSampleDBException {

  ejbSampleDBObj dbObj = new ejbSampleDBObj();
  dbObj.userid = userid;

  try {
    dbStmt.setString            (1, dbObj.userid);
    dbStmt.execute();

    dbObj.retCode = dbStmt.getString(8).trim();

    if (dbObj.retCode.equals("0")) {
      dbObj.lastName       = (dbStmt.getString(2)).trim();

      dbObj.firstName      = (dbStmt.getString(3)).trim();
      dbObj.middleName     = (dbStmt.getString(4)).trim();
      dbObj.employer       = (dbStmt.getString(5)).trim();
      dbObj.allowances     = (dbStmt.getString(6)).trim();
      dbObj.addWithholding = (dbStmt.getString(7)).trim();
    }
```

```
  } catch (SQLException se) {
```

> **Note:** If the database connection is lost while a servlet instance is still alive, attempting to use the connection will raise the `SQLException` exception. If the connection is not re-established, all requests sent to this servlet instance will fail. Additional error checking should be added to check for connection failure and perform appropriate exception handling. This will make your application more robust.

```
    throw new ejbSampleDBException(se.toString());
  } catch (Throwable e) {
    throw new ejbSampleDBException(e.toString());
  }
  return dbObj;
} // getSampleData


//  Get database connection
private void getDBConnection() throws Throwable {
  try { // open the database connection
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    dbConn = DriverManager.getConnection(
             "jdbc:oracle:oci8:@" + dbServer, "scott", "tiger");
  } catch (Throwable e) {
    logger.println("Exception in getDBConnection: " + e.toString());
    throw e;
  }

  try { // set up prepare statement
    dbStmt =
      dbConn.prepareCall("begin GET_PERFSAMPLE_DATA(?,?,?,?,?,?,?,?); end;");
    dbStmt.registerOutParameter (2, Types.VARCHAR);
    dbStmt.registerOutParameter (3, Types.VARCHAR);
    dbStmt.registerOutParameter (4, Types.VARCHAR);
    dbStmt.registerOutParameter (5, Types.VARCHAR);
    dbStmt.registerOutParameter (6, Types.VARCHAR);
    dbStmt.registerOutParameter (7, Types.VARCHAR);
    dbStmt.registerOutParameter (8, Types.VARCHAR);
```

```
    } catch (Throwable e) {
      logger.println("Exception in prepareCall: " + e.toString());
      throw e;
    }
  } // getDBConnection
} // class
```

This example uses the database object defined Example 2–5.

Example 2–5 The EJB sample database object

```
package ejbSample;

public final class ejbSampleDBObj implements java.io.Serializable {
  public String  userid;
  public String  lastName;
  public String  firstName;
  public String  middleName;
  public String  employer;
  public String  allowances;
  public String  addWithholding;
  public String  retCode;
}
```

In Example 2–6 we see the JServlet code that invokes this EJB. The servlet code does not implement the `SingleThreadModel` interface, so there will only be one instance per process. We use JNDI's synchronization ability to guarantee thread safety. This allows us to get the JNDI home only once. This results in a performance increase because getting the JNDI home is an expensive operation.

The `init()` and `destroy()` methods are only called once. Since this servlet does not implement the SingleThreadModel, these methods will be called only once per process no matter how many instances are created for this servlet. The `init()` method acquires the EJB application context and the context is closed in the `destroy()` method. All threads invoking the servlet will share this context when getting an EJB from the EJB server.

To invoke the EJB with the servlet, the servlet must include the following JAR files in its CLASSPATH:

- $ORACLE_HOME/ows/apps/ejb/<*ejbSample*>/_client.jar
- $ORACLE_HOME/ows/apps/ejb/<*ejbSample*>/_application.jar

where <*ejbSample*> is the name of the EJB application.

Further information on invoking the JServlet application or changing the code so it works on your system, see "Example Notes" on page 2-8.

Example 2–6 JServlet that invokes ejbSampleDB

```
import ejbSample.*;

import java.io.*;
import java.lang.*;
import java.sql.*;
import java.util.*;
import javax.ejb.CreateException;
import javax.ejb.RemoveException;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import javax.servlet.*;
import javax.servlet.http.*;

public class JSampleEJB extends HttpServlet {
  // String constants
  private final static String COMPONENT_NAME = "JSampleEJB";
  private final static String USERID         = "userid";
  private final static String SAMPLE_EJB     = "ejbSample/ejbSampleDB";

  // Instance variables
  private Context initialContext  = null;
  private ejbSampleDBHome sampleHome = null;

  // The init() method is called when the servlet instance is first created.
  public void init(ServletConfig config) throws ServletException {
    super.init(config);

    // Get configuration information.
    String url = config.getInitParameter("RMProxyURL");
    if (url == null) { url = "oas:///"; }

    // Get initial naming context and the application context for
    // the ejbSample application.  The context for this EJB will
    // be shared by all threads in this servlet.
    try {
      initialContext = (Context) (new InitialContext()).lookup(url);

      sampleHome = (ejbSampleDBHome)
        PortableRemoteObject.narrow(initialContext.lookup(SAMPLE_EJB),
        ejbSampleDBHome.class);
```

```
  } catch (Throwable e) {
    throw new ServletException(e.toString());
  } // try
} // init


// The destroy() method is called when the servlet instance is shutdown.
public void destroy() {
  // Release the initial naming context and the ejbSample application
  // instance context.  Closing the initial context will close all
  // contexts under the initial context.
  if (initialContext != null) {
    try {
      initialContext.close();
    } catch (Throwable e) {
      getServletContext().log("Exception in closing EJB context:" +
        e.toString());
    } // try
  } // if
} // destroy


// The doGet() method is the entry function for every request.
public void doGet (HttpServletRequest httpRequest,
                   HttpServletResponse httpResponse)
throws ServletException, IOException {
  ejbSampleDBRemote sampleEJB = null;
  ejbSampleDBObj sampleObj = null;
  PrintWriter httpOut;

  // Initialize HTTP response header
  httpResponse.setContentType("text/html");
  httpOut = httpResponse.getWriter();
  httpOut.println("<html>");
  httpOut.println("<head><title>" + COMPONENT_NAME + "</title></head>");

  // Get user id parameter
  String userid = httpRequest.getParameter(USERID);
  if (userid == null) {
    pgError(httpOut, "Unable to retrieve user information, " +
      USERID + " is null");
    return;
  } // if
```

```
// Get an EJB instance
try {
  sampleEJB = (ejbSampleDBRemote) sampleHome.create();
} catch (java.rmi.RemoteException re) {
```

Note: If the EJB server is down, the create() method will throw a `RemoteException` exception. If the servlet does not acquire a new EJB application context, all subsequent requests sent to this servlet will fail with the same error.

```
  throw new ServletException("EJB Create Exception: " + re.toString());
} catch (Throwable ce) {
  throw new ServletException(ce.toString());
} // try

// Invoke the EJB method and release the EJB instance upon completion.
try {
  sampleObj = sampleEJB.getSampleData(userid);
  if (sampleObj.retCode.equals("0")) {
    pgOut(httpOut, sampleObj);
  } else if (sampleObj.retCode.equals("100")) {
    pgError(httpOut, "Record not found for " + userid);
  } else {
    pgError(httpOut, "Unable to retrieve record for " + userid +
                     " (SQLERROR " + sampleObj.retCode);
  }
} catch (Throwable e) {
  throw new ServletException("EJB Invoke Exception: " + e.toString());
} finally {
  try {
    if (sampleEJB != null) {
       sampleEJB.remove(); }
  } catch (Throwable e) {
    getServletContext().log("EJB Remove Exception: " + e.toString());
  } // inner try
} // outer try

} // doGet
```

```
  // Print HTML output page
  private void pgOut (PrintWriter httpOut, ejbSampleDBObj sampleObj) {
    String userName = sampleObj.firstName + " " + sampleObj.middleName + " " +
                      sampleObj.lastName;
    httpOut.println("<body>");
    httpOut.println("<h1>User: <font color=blue>" +
                    sampleObj.userid + "</font> </h1>");
    httpOut.println("<h1>Name: <font color=blue>" +
                    userName + "</font> </h1>");
    httpOut.println("<h1>Employer: <font color=blue>" +
                    sampleObj.employer + "</font> </h1>");
    httpOut.println("<h1>Number of Allowances: <font color=blue>" +
                    sampleObj.allowances + "</font> </h1>");
    httpOut.println("<h1>Additional Amount of Withholding: <font color=blue>" +
                    sampleObj.addWithholding + "</font> </h1>");
    httpOut.println("</body>");
  }


  //
  // Print HTML error page
  private void pgError (PrintWriter httpOut, String errmsg) {
    httpOut.println("<body>");
    httpOut.println("<h1> <font color=red> ERROR: " + errmsg +
                   "</font> </h1>");
    httpOut.println("<hr>");
    httpOut.println("</body>");
  } // pgError
} // JSampleEJB class
```

PL/SQL Applications

PL/SQL cartridge users should consider the following topics when developing their applications.

Database Access Descriptors (DADs)

Each PL/SQL cartridge server, on startup, attempts to connect to the database specified by each Database Access Descriptor (DAD) listed in the Oracle Application Server configuration. To save time, you should restrict the number of DADs on each Oracle Application Server node, listing only those needed by the applications on that node. See the *Administration Guide* for more information on creating and managing DADs.

Nested Tables

PL/SQL provides the ability to create tables. To build PL/SQL tables, you build a table that gives the data type of the table, as well as the index of the table. The index of the table is the binary integer ranging from -2147483647 to +2147483647. This table index option is known as sparsity, and allows meaningful index numbers such as customer numbers, employee number, or other useful index keys. Use PL/SQL tables to process large amounts of data.

PL/SQL provides TABLE and VARRAY (variable size array) collection types. The TABLE collection type is called a nested table. Nested tables are unlimited in size and can be sparse, which means that elements within the nested table can be deleted using the DELETE procedure. Variable size arrays have a maximum size and maintain their order and subscript when stored in the database. Nested table data is stored in a system table that is associated with the nested table. Variable size arrays are suited for batch operations in which the application processes the data in batch array style. Nested tables make for efficient queries by storing the nested table in a storage table, where each element maps to a row in the storage table.

3

Sizing and Configuration

This chapter provides tips for sizing and configurations which can improve your ability to meet performance goals. It also provides an overview of performance factors, such as CPU and I/O issues, for your operating system and your hardware.

Contents

Installation Requirements

The following table describes the minimum hardware requirements for an Oracle Application Server installation.

Table 3–1 Hardware requirements

Hardware Item	Required
CPU	A Sun UltraSPARC II processor or Pentium Pro at 200 Mhz
Memory	128 MB
Disk Space	500MB
Swap Space	256 MB
Network Connection	100 Mbps per second

Your actual requirements depend on your web applications and the number of users that are actively using the site during peak hours. See the *Oracle Application Server Installation Guide* of your respective platform for additional information about the hardware and software requirements.

Sizing your Hardware and Resources

In addition to the minimum installation recommendations, your hardware resources need to be adequate for the requirements of your specific applications. To avoid hardware-related performance bottlenecks, each hardware component should operate at no more than 80% of capacity.

Processor and memory resources in particular should be more than adequate to handle the maximum traffic that your network connections can handle. If your network becomes a bottleneck, you can upgrade to faster network interface cards, or install multiple network interface cards on each machine.

For smaller workloads, a single Oracle Application Server node meeting the hardware recommendations can run listeners, cartridge servers, and databases.

See “Monitoring Performance Statistics” on page 5-1 to learn how to assess your Oracle Application Server performance.

Determining User Population

The amount of hardware resources required varies based on your individual application. A common mistake is to use resource estimates that do not incorporate user "think time" and network latencies. If you have deployed an application using an older version of Oracle Application Server, you probably have some idea of the relationship between the number of potential users and the number of actual concurrent users.

Determining CPU Requirements

As with memory resources, the amount of CPU resources varies based on the application. You can assume a base amount of 20msec of CPU time per request on a 336 Mhz Solaris system (10 for CWeb). For example, if you require your system to handle 200 requests a second, then you need 4 processors to meet the requirement.

```
Total CPUs= (200 requests/sec X 20msec CPU time per request)/1000 (msec/sec)
Total CPUs= 4 CPUs
```

Determining Memory Requirements

A way to determine the amount of memory required on an Oracle Application Server is to determine the following:

Memory for Non-OAS Software and Operating System

Fixed Memory Cost

Variable Memory Requirements

Number of Concurrent Executing Users

Cost per JVM for Java- based applications

Memory for Non-OAS Software and Operating System

When you use operating system tools such as **ps -elf** or **ps -aux** on UNIX to look at the size of non-Oracle Application Server processes, you may notice that the processes seem relatively large. To interpret the statistics shown, you must determine how much of the process size is attributable to shared memory, heap, and executable stack, and how much is the actual amount of memory the given process consumes.

Refer to your operating system hardware and software documentation for more information on measuring and tuning operating system memory usage.

Fixed Memory Cost

In sizing systems to improve your performance, you need to take into consideration the memory consumption of processes other than the cartridge processes plus any operating system overhead. On Solaris 2.6 system with 1 GB of RAM, a test of concurrent 50 users (not including think time) the memory consuption for processes other than the cartridge server processes was 50.5 MB (90.9 MB virtual memory).

The following table provides an example of the fixed memory consumption of the various Oracle Application Server processes:

Table 3–2 Fixed memory cost of running Oracle Application Server processes

Process	Resident Memory (MB)	Virtual Memory (MB)
oassrv	14	28
oasorb	4.5	5.9
wrksf	15	26
otsfacsrv	7	14
oraweb	10	17
FIXED COST	**50.5**	**90.9**

Note: The oraweb process does not include memory for the admin and node manager listeners as these will be infrequently accessed.

In an idle system, where memory resources are freely available, your operating system statistics may indicate that the resident memory usage is closer to the virtual size. As users place more load on the system, the operating system reclaims unneeded memory from these processes and the amount of resident memory they consume decreases. If you are monitoring your own system, take snapshots of fixed processes at varying usage levels.

You can monitor memory usage of a process in UNIX by using the **ps** or **top** command.

Variable Memory Requirements

The amount of memory required per user varies on your individual application and its type (JServlet or PL/SQL). However, as a starting point, you can assume that each active user consumes a t least 150K to 200K for Java or PL/SQL applications, and 100K for CWeb applications, plus the size of the server processes.

In addition, for lightly loaded systems, assume one process per CPU. For heavily loaded systems, assume two to three processes per CPU. For Java applications, the base process is approximately 12-15 MB.

Table 3–3 Resources required for various applications

Application	Users (no think time)	Total Resident Memory (MB)	Variable Memory per thread (KB)	CPU time per request (msec)
JServlet	100	15	150	~20
PL/SQL	100	15	200	~20
CWeb (no database)	100	9.2	103	~10

Note: Total Resident Memory is equal to the wrks resident memory when the system is configured with one server process and one instance.

The amount of memory required for your application also depends on the following:

- Size of your own applications.
- Amount of data you want to cache if using the JServlet session cache.
- Number of server processes and threads created by OAS to service your request.

Number of Concurrent Executing Users

In determining memory requirements, you also need to consider the number of concurrent executing users (not the total user population) times the cost per user.

The following table provides an example of the impact of think time and service time on the concurrency and resulting performance of a system:

Table 3–4 Concurrent Executing Users

Number of Users	Think Time (sec)	Service Time (sec)	Range of users	Average response Time (sec)	Requests per Second (throughput)	CPU Utilization (%)
100	0	1	100	3.5	28	100
100	1	1	40-100	2.3	29	97
100	10	1	30-40	1.2	9	58
100	10	2	40-60	2.2	8	77

Service Time in Seconds - elapsed time to complete the operation measured for a single user.

Range of Users - the number of users measured on the server, taken in snapshots through the measurement.

Average Response Time - response time measured at the client under load.

Requests per Second - throughput for the entire server.

Cost per JVM for Java- based applications

Although the Policy Manager controls the number of server processes based on the load and available resources on the system, you can estimate 1-2 servers per application per CPU. The exact number will vary based on a variety of factors.

Java applications consume more memory than other types of applications. The typical size of a Java Virtual Machine (JVM) is 9-12 MB of resident memory.

Performance Factors

Performance problems tend to be interconnected rather than isolated. For example, cache problems may show up as issues of CPU, memory, or I/O. This section provides an overview of the major factors that influence Oracle Application Server performance:

- Insufficient Memory
- Insufficient CPU Resources
- Insufficient I/O
- Network Constraints
- Software Constraints

Insufficient Memory

Some memory problems appear to be I/O problems. There are two kinds of memory requirements: Oracle Application Server and system. For more information about improving memory requirements for OAS, see Determining Memory Requirements.

Oracle Application Server memory requirements impact the system requirements. Memory problems may be the cause of all the paging and swapping that goes on in the machine. Make sure that your system does not start swapping and paging. The whole system should be able to run within the limitations set by internal memory.

On the system level you can trim the number of processes and/or the amount of memory each process uses. You can also identify which processes are using the most memory.

Insufficient CPU Resources

In a CPU-bound system, the CPU resource is completely used up, service time is too high and you want to achieve more. Alternatively, you have too much idle time, want to achieve more, and the CPU is not completely used up. There is room to do more: you need to determine why so much time is spent waiting.

To diagnose insufficient CPU, you must check CPU utilization by your entire system, not only utilization by Oracle Server Application processes. At the beginning of a workday, for example, the mail system may consume a very high amount of the available CPU, while employees check their messages. Later in the day, the mail system may be much less heavily used, and its CPU utilization will drop accordingly.

Workload is a very important factor when evaluating your system's level of CPU utilization. During peak workload hours, 90% CPU utilization with 10% idle and waiting time may be understandable and acceptable. Thirty percent utilization at a time of low workload may also be understandable. However, if your system shows high utilization at normal workload, there is not room for peak workload. You have a CPU problem if idle time and time waiting for I/O are both close to zero (less than 5%) at a normal or low workload.

Insufficient I/O

Be sure to evenly distribute I/O across disks and channels.

- channel bandwidth: number of I/O channels
- device bandwidth: number of disks
- device latency: latency will be part of your wait time

I/O problems may result from the limitations of your hardware configuration. Your system needs enough disks and SCSI busses to support the transaction throughput you desire. You can evaluate the configuration by figuring the number of messages your disks and busses can support, and comparing that to the number of messages required by your peak workload.

If the response time of an I/O becomes too high, the most common problem is that the wait time has gone up (response time = service time + wait time). If wait time goes up, it means that there are too many I/O requests for this device. If service time goes up, this normally goes hand in hand with a larger I/O, so you write more bytes.

The different background processes perform different kinds of I/O, and each process have different I/O characteristics. Some I/O processes read and write in the block size of the database, some read and write in larger chunks. If service time is too high, stripe the file over different devices.

Network Constraints

Network constraints are similar to I/O constraints. You need to consider:

- network bandwidth: each transaction requires that a certain number of packets be sent over the network. If you know the number of packets required for one transaction, you can compare that to the bandwidth to determine whether your system is capable of supporting the desired workload.
- message rates: you can reduce the number of packets on the network by batching them up rather than sending lots of small packets.
- transmission time

As the number of users and the demand rises, the network can sometimes quietly become the bottleneck in an application. You may be spending a lot of time waiting for network availability. Use available operating system tools to see how busy your network is.

Software Constraints

Operating system software determines:

- the maximum number of processes you can support
- the maximum number of processes you can connect

Before you can effectively tune Oracle Application Server, you must ensure that the operating system is at its peak performance. Work closely with the hardware/software system administrators to ensure that Oracle Application Server is allocated the proper operating system resources.

Note: On NT systems there are no pre-set or configurable maximum numbers of processes that can be supported or connected.

See Also: Operating system tuning is different for every platform. Refer to your operating-system hardware/software documentation as well as your Oracle operating system-specific documentation for more information.

4

Tuning Oracle Application Server Components and Parameters

This chapter discusses configurations and parameters which can improve performance.

Contents

Tuning Processes

File Descriptors per Process

Make sure that the limit on file descriptors per process is set to the maximum (1024) before starting Oracle Application Server. This allows your listeners to maintain as many open connections as possible.

Using csh

To find the current limit, enter:

```
limit descriptors
```

To set the file descriptors to the maximum, enter:

```
unlimit descriptors
```

Using ksh

To find the current limit, enter:

```
ulimit -n
```

To set the file descriptors to the maximum, enter:

```
ulimit -n 1024
```

Refer to your operating system for the proper command.

Distributing the Authentication Server Processes

The Authentication Server component of Oracle Application Server handles security on behalf of dispatchers and cartridges/components. When a dispatcher receives an HTTP request requiring access to a protected virtual path, it contacts the Authentication Server and waits for its response to determine whether the request can be filled.

You can run the Authentication Server on machines other than the primary node, and you can run multiple copies of the Authentication Server. The main reasons for doing this are performance and reliability. Consider the following points:

- If the primary node is running many processes and resources on the node are scarce, you might get better performance if you move the Authentication Server to a less busy machine.
- If you are using the Oracle database server to authenticate clients, you can improve performance if you move the Authentication Server to the same machine as the database.
- If you have only one Authentication Server and several clients are requesting authentication, then the requests are queued. You can improve this bottleneck by running multiple copies of the Authentication Server.
- If you are running multiple copies of the Authentication Server on different machines and one machine fails, clients can still access the remaining Authentication Servers.

The services for the Authentication Server are:

- **ORAWEB40_wrbasrv** - the Authentication Broker service
- **ORAWEB40_wrbahsrv** - the Authentication Provider service

Installing the Authentication Server on Remote Nodes

To run the Authentication Server on a remote node, install Oracle Application Server on the remote node. During installation, select the multi-node option, then select “remote”, and then select “WRB” when prompted to select the components to install on the remote node.

During installation of the Web request Broker (WRB) on the remote node, you need to provide the name of the primary node.

Tuning Listeners

This section contains suggestions and recommendations for improving the performance of listeners.

Multiple Listeners

Each listener can accommodate a maximum number of concurrent connections. This number varies based on operating system restrictions. For specific numbers, see the appropriate documentation for your platform.

To distribute the request load on a site, create multiple listeners on the site, each listening on a different TCP port. On multiprocessor machines, this increases the throughput of your Oracle Application Server site by allowing more concurrent connections.

Files in a Directory

Avoid storing a large number of files in a directory served by a listener. The more files that are stored in a directory, the longer it takes to search the directory for a requested file. To store a large number of files, distribute the files among several directories, minimizing the size of each.

Oracle recommends that you store no more than 1000 files in each directory.

Tuning Web Listeners

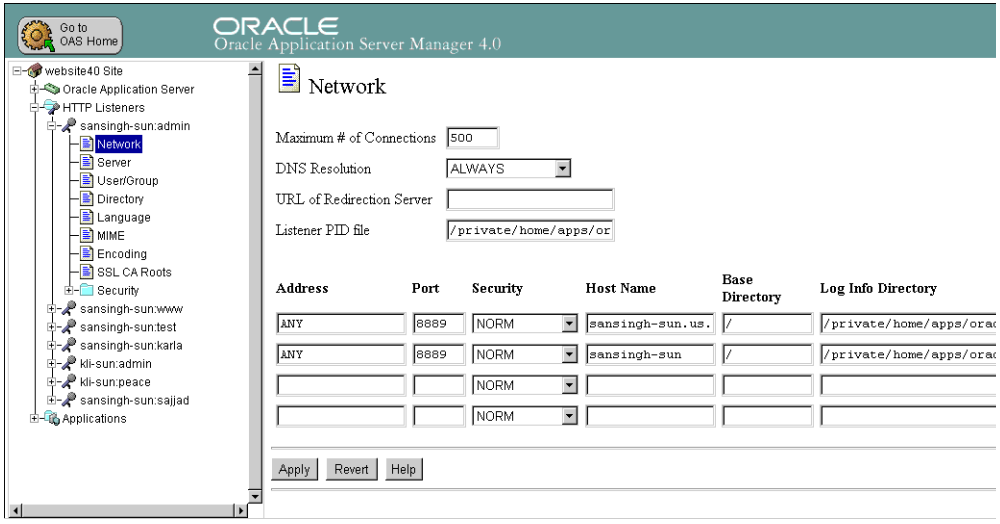
Consider the following performance suggestions:

Disable DNS Resolution

For Web Listeners that do not support domain-based restriction, you can reduce request latency by disabling DNS resolution. To do this:

1. Connect to the Oracle Application Server Welcome page and click on the OAS Manager icon.
2. Expand the tree structure under the site you want to configure ("website40" by default).
3. Click on HTTP Listeners in the left frame.
4. Expand the tree structure under the **www** listener you want to configure.
5. Click on Network.

Figure 4–1 Network Form



6. From the DNS Resolution pull-down menu, choose NEVER.
7. Click Apply.

> **Note:** If you disable DNS resolution, the listener will be unable to perform domain-based restriction. If the listener must support this kind of restriction, choosing LAZY or LAZY_WITH_CGI from the DNS Resolution pull-down menu is a good compromise.

8. Reload the **www** listener.
   a. Click on HTTP Listeners in the left frame.
   b. Select the modified **www** listener and click the Reload button  .

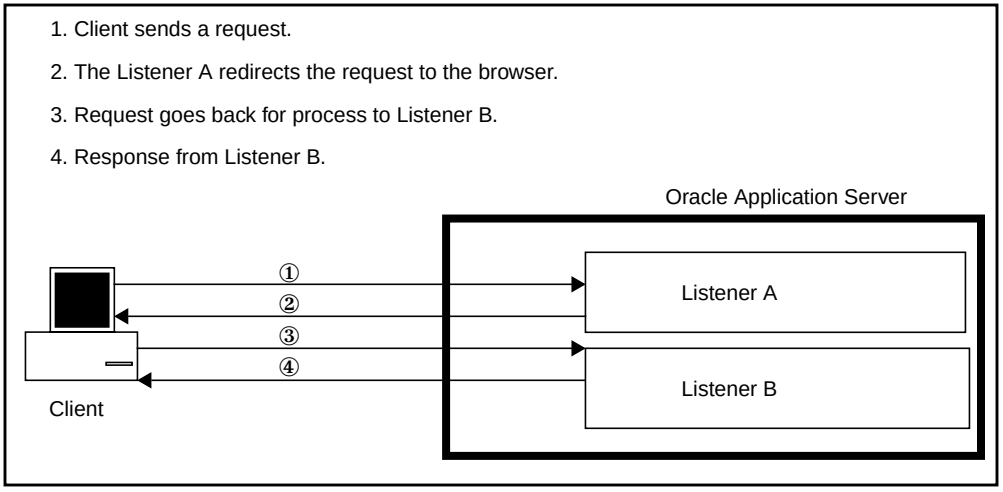
Adjust the Maximum Number of Connections and Configure Automatic Redirection

You can specify a maximum number of requests that an Oracle Web Listener can concurrently handle. The default value is 500 for both NT and UNIX versions. The maximum number of connections on NT is 2000. This number is limited by system resources. The maximum for UNIX is 700, assuming your operating system is set up to have 1024 file descriptors per process. (For more information, see "File Descriptors per Process" on page 4-2.)

You can also specify the URL of another listener to which any requests in excess of this number should be redirected. For example, if a listener's maximum number of requests is set to 100, the 101st request is sent back to the browser with a redirection status and the URL of the second listener. The browser interprets this status and sends a request to the second listener. This operation is transparent to the user of the browser.

Listener redirection is not free. The response time for an individual user may actually increase due to the extra network overhead. Because a user's request must travel from the client, to listener A, back to the browser, and then to listener B, it incurs two additional network hops.

Figure 4–2 Listener redirection



Listener redirection works best when the cost of the redirection is amortized across multiple subsequent accesses. Once the browser receives the URL for the second listener, it caches this information and uses it in resolving subsequent requests for URLs with paths relative to the initial page.

Because of the additional network overhead, you should only use redirection if the throughput limits of the listener have been reached. You can use **sar**, or one of the other system utilities discussed in the previous chapter to determine whether this is the case. If the listener can handle more requests, but is refusing them, increase the "Maximum # of Connections" accordingly. If it is already at its maximum for the platform Oracle Application Server is running, or if you have determined that the throughput limit has been reached, configure the listener to redirect additional requests to a listener on a different host / port.

Automatic redirection will not improve performance (and can even worsen performance) under the following situations:

- The load on the system does not exceed the capacity of the initial listener.
- The requests are for stand-alone static pages or small cartridges, where each request requires a redirection.

To adjust the maximum number of connections allowed concurrently, and if necessary to configure automatic redirection:

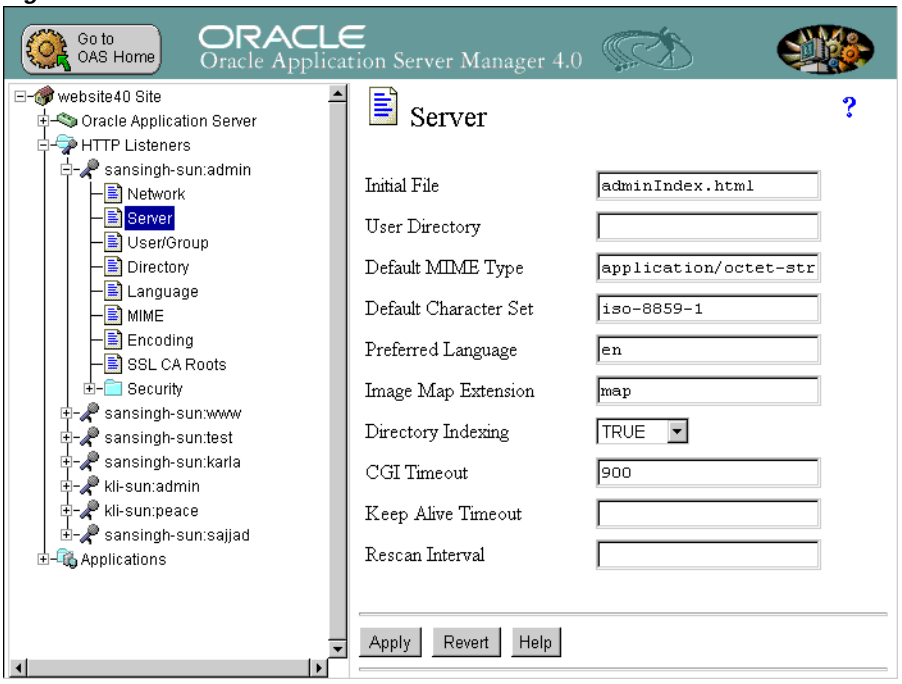
1. Connect to the Oracle Application Server Welcome page and click on the OAS Manager icon.
2. Expand the tree structure under the site you want to configure (“website40” by default).
3. Expand the tree structure under the **www** listener you want to configure.
4. Click on Network.
5. In the Maximum # of Connections text field, enter the number of connections your system can handle simultaneously (or the maximum permitted).
6. If automatic redirection is necessary, then in the URL of Redirection Server text field, enter the URL of the listener you want to handle redirected requests from this listener.
7. Click Apply.
8. Reload the **www** listener.
   a. Click on HTTP Listeners in the left frame.
   b. Select the modified **www** listener and click the Reload button  .

Minimize Directory Rescans

By default, an Oracle Web Listener rescans a directory looking for new, changed, and deleted files every time the directory is accessed. Because most directories do not change frequently, it is usually acceptable to rescan directories only once per hour, eliminating the per-access rescan overhead. To adjust the Rescan Interval:

1. Connect to the Oracle Application Server Welcome page and click on the OAS Manager icon.
2. Expand the tree structure under the site you want to configure ("website40" by default).
3. Click on HTTP Listeners in the left frame.
4. Expand the tree structure under the **www** listener you want to configure.
5. Click on Server.

Figure 4–3 Server Form



6. In the Rescan Interval text field, enter 3600 for the number of seconds (one hour).

7. Click Apply.
8. Reload the **www** listener.
   a. Click on the HTTP Listeners in the left frame.
   b. Select the modified **www** listener and click the Reload button  .

Keep Alive Timeout

This parameter allows you to determine how long the listener waits before timing out a connection. If this value is low, the listener will be disconnecting and reconnecting frequently. This should be set so that the listener waits a reasonable amount to time before closing a connection. To configure the Keep Alive Timeout:

1. Connect to the Oracle Application Server Welcome page and click on the OAS Manager icon.
2. Expand the tree structure under the site you want to configure ("website40" by default).
3. Click on HTTP Listeners in the left frame.
4. Expand the tree structure under the **www** listener you want to configure.
5. Click on Server.
6. In the Keep Alive Timeout field, enter the number of seconds.

   (The default is 10 seconds.)
7. Click Apply.
8. Reload the **www** listener.
   a. Click on the HTTP Listeners in the left frame.
   b. Select the modified **www** listener and click the Reload button  .

Tuning Cartridges

Load Balancing Schemes

Oracle Application Server provides two different load balancing schemes for applications, priority-based, and minimum and maximum number of cartridge instances.

Priority Priority tuning manages and allocates your system resources automatically based on the priority level you set for your applications and cartridges. The priority levels are High, Medium, Low, and Discretionary.

Min/Max Min / Max tuning is a manual task requiring you to set minimum and maximum number of instances at the application and cartridge levels.

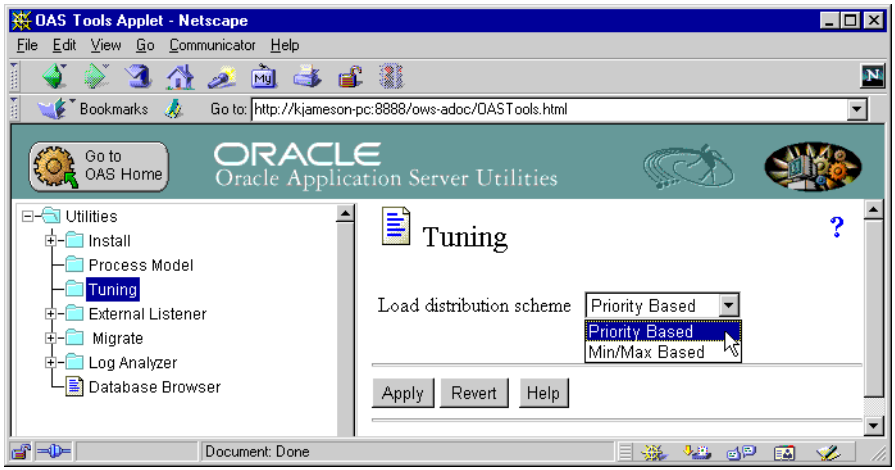
(By default, Oracle Application Server uses priority-based load balancing.)

Changing the Load Balancing Scheme

By default, Oracle Application Server runs in the priority-based scheme, to change your load-balancing scheme:

1. Connect to the Oracle Application Server Welcome page and click on the OAS Utilities icon.
2. Expand the Utilities folder and click on Tuning.
3. Select the load balancing scheme from the Load distribution scheme pull-down menu.

Figure 4–4 Choosing a load distribution scheme



4. Restart Oracle Application Server.

Priority-Based Load Balancing

Note: The instructions in this section apply only to cartridges that are both globally thread-safe and instance-context thread-safe.

Introduced in Oracle Application Server 4.0.8, priority-based tuning provides automatic load balancing for your applications and cartridges. This is the most efficient and labor-saving method for handling load balancing. Unless you are familiar with versions of Oracle Application Server prior to 4.0.8 (where Min / Max was the only way to load balance), Oracle recommends that you use priority-based load balancing.

Each application and cartridge is automatically set to Medium. You should adjust each cartridge or application depending on your needs.

Set the load balancing priority classification at the application or cartridge level to High, Medium, Low, or Discretionary. The number of processes, threads, and instances is automatically determined based on the request load and priority level of the application and components.

Note: Cartridges inherit their priority setting from their application when an explicit cartridge priority is not set.

Requests from high priority applications or cartridges receive preferential treatment over requests from lower priority applications or cartridges. Requests for lower priority applications or cartridges are serviced when sufficient resources exist.

Table 4–1 explains how requests for applications and cartridges are prioritized depending on their respective priority level.

Table 4–1 Priority levels

Priority	Description
High	The system will try to allocate, whenever possible, more resources and give preference to service these requests over lower priority Cartridges (i.e. Medium, Low or Discretionary). A similar reasoning is applied if you choose Medium or Low priority.
Medium	The system default setting, these requests are serviced if no high priority requests are waiting for resources.
Low	Requests are only serviced when adequate resources are available.
Discretionary	The system only services these requests if there are enough resources available. Moreover, these cartridges could be shutdown without being serviced if resources are still required by High, Medium or Low priority Cartridges. The main difference between Low and Discretionary Cartridges, is that Low priority Cartridges cannot be shutdown.

For instructions on selecting a process model, see "Changing the Load Balancing Scheme".

Min/Max Based Load Balancing

Establishing the Min / Max settings allows you to manually set configuration parameters for your applications and cartridges. You can set upper and lower bounds for instances and threads to balance cartridge servers, instances, and threads based on CPU utilization.

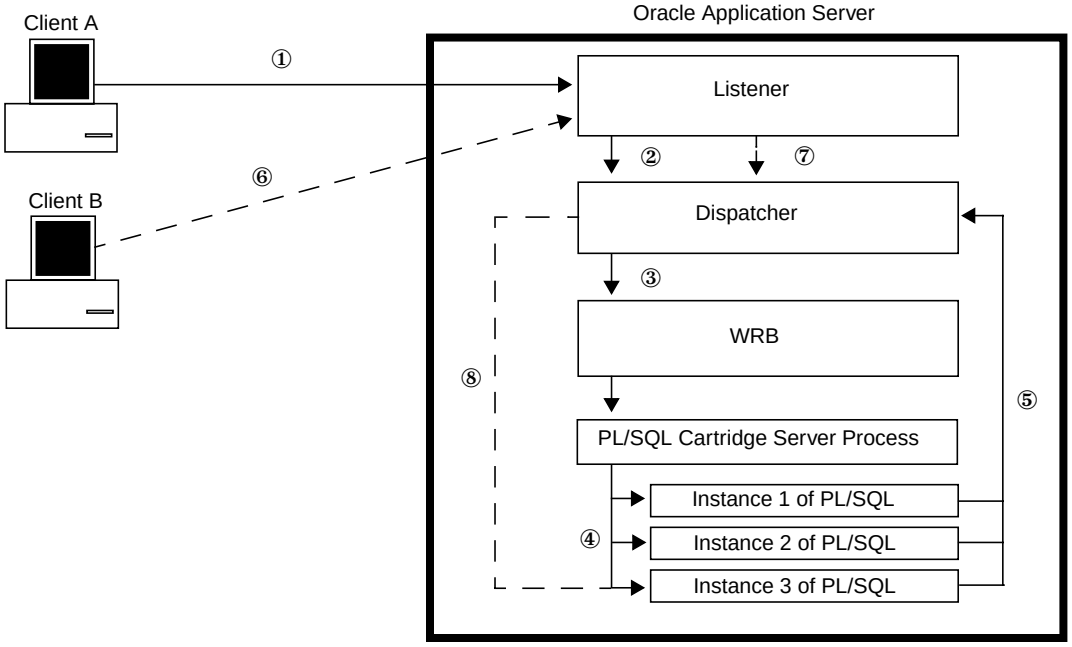
Oracle Application Server uses Cartridge Server Processes (**wrks**), which improve performance by invoking multiple threads / instances of a particular cartridge. Figure 4–5 shows a flowchart of cartridge instance lifecycle:

1. A client sends a request for a cartridge to the Listener.
2. The Listener sees that the request is for a cartridge, and sends it to the Dispatcher.
3. If the Dispatcher knows of no free cartridge instances for that cartridge, it sends the request to the Web Request Broker (WRB).
4. The WRB then directs one or more cartridge server processes to allocate cartridge instances (also known as “threads”). The WRB tries to ensure that each cartridge server process allocates approximately the same number of cartridge instances for a particular cartridge for each node.
5. Each cartridge server process creates the appropriate number of cartridge instances for its node.
6. The cartridge instances are then registered with the Dispatcher, so that the Dispatcher can direct requests to them.

   When subsequent requests for the cartridge come in, the Dispatcher sends the requests to unoccupied cartridge instances.

Figure 4–5 Cartridge instance flowchart

1. Client A sends a request for the PL/SQL cartridge.
2. The Listener forwards the request to the Dispatcher.
3. The Dispatcher knows of no free instances of the PL/SQL cartridge, forwards request to the WRB.
4. The WRB instructs a PL/SQL cartridge server process to launch instances of the PL/SQL cartridge.
5. The new instances of the PL/SQLcartridge are registered with the Dispatcher.
6. Client B sends a request for the PL/SQL cartridge.
7. The Listener forwards the request to the Dispatcher.
8. The Dispatcher sends the request directly to an available instance of the PL/SQL cartridge.



Tuning Logging

CLF and XLF Logging

Most web sites track the site access statistics by collecting Common Log Format (CLF) data, which can be processed and analyzed with a wide range of tools.

In Oracle Application Server, CLF logging data is collected by default. You can configure site access logging from the Oracle Application Server Manager in the XLF form under logging. The XLF logging form also provides the ability to specify which site access statistics you want to collect (CLF is the default). OAS will produce a single merged log for all active listeners in an OAS site.

You can substantially improve your listener performance, however, by opting to have each listener log its own access statistics (both CLF and XLF) separately as follows:

1. Turn XLF logging to OFF in the logging form by using the Oracle Application Server Manager.
2. Shut down Oracle Application Server.
3. Edit the listener configuration file for each port number on which you have a listener that you want to access logging:

   ```
   cd $ORAWEB_ADMIN/$ORAWEB_SITE/httpd_<host-name>/<listener-name>
   <edit> sv<listenername>.cfg
   ```

4. Check that a line such as the following is found in the [MultiPort] section of the file.

   ```
   ANY    <port-number>    NORM    <host-name>.<domain>/    <log-dir>    NONE
   ```

5. Add the [ConnectionLogs] section in the file if there is none. Add one just above the [Server] section, and a line which says what to store in what file:

   ```
   <clf-log-file> CLF {clf}
   ```

6. Restart the Oracle Application Server.

The above steps improve performance significantly even on OAS sites with a single listener.

Example:

1. Turn XLF logging to OFF in the logging form by using the Oracle Application Server Manager.
   a. From the Welcome page, click on OAS Manager.
   b. Expand the site to be edited.
   c. Expand Oracle Application Server.
   d. Expand Logging and click on XLF.
   e. In the Logging field, choose OFF.
   f. Click Apply.
2. Shut down Oracle Application Server.
3. Edit the listener configuration file:

   ```
   cd $ORAWEB_ADMIN/$ORAWEB_SITE/httpd_myhost/www
   vi svwww.cfg
   ```

4. Check for the following line under [MultiPort] section.

   ```
   ANY    80    NORM    myhost.mysite.com/    /var/log/oas/www    NONE
   ```

   If you do not find the line, add a line specifying the directory in which you want your log files.

5. Add the [ConnectionLogs] section just above the [Server] section:

   ```
   ;
   [ConnectionLogs]
   clf.log CLF {clf}
   xlf.log XLF {time cs-method cs-uri}
   ;
   [Server] ...
   ```

   This results in the creation of a standard CLF log file **clf.log** and a special purpose **xlf.log** being created in the directory in the fourth field of the [MultiPort] entries.

   The XLF line shows how additional extended log format (XLF) data can be collected if desired. For more information about XLF formats, see the *Oracle Application Server Administration Guide*.

6. Restart the Oracle Application Server.

System Logging

To minimize request latency on a stable system, you can specify that the System Logging Service log messages for only the most severe errors. You may want to use more extended logging for debugging purposes during your development process, but since log messages translate into physical I/O, it is recommended to use only logging errors for productions systems.

1. From the Welcome page, click on OAS Manager.
2. Expand the site to be edited.
3. Expand Oracle Application Server.
4. Expand Logging.
5. Click on System.
6. In the Logging Directory text field, enter the full path of a directory on a disk that is not used to store Web or application data.
7. In the Logging File field, enter the name of the logging file.
8. From the Severity Level pull-down menu, choose 1.
9. Click Apply.
10. Select Oracle Application Server in the left frame and click Reload.

Monitoring Error and Log Files

Excessive or repeated errors consume system resources and slow response time. Regular checking of files helps detect inefficiencies in Oracle Application Server configuration.

For example, a repeated HTTP Error 500 might be caused by an imagemap tag with undefined areas; every client that clicks in one of these areas generates the error. You can eliminate the problem by eliminating the “holes” in the imagemap tag. Another example is a repeated HTTP Error 404, which is usually caused by a broken hypertext link.

Regular monitoring of log files provides a good understanding of how clients access your site. You can use this information to optimize allocation of hardware and server resources. You can use tools such as WebTrends for detailed site usage monitoring.

Do not allow your log files to grow without limit. When a log file reaches a certain size, archive the file and start a new log file. This avoids the overhead of writing to a large file.

To minimize disk access contention, you can also specify that system messages be logged to a file on a disk that is not used to store Web or application data. The default for OAS is to batch log messages and write them to disk in groups of 100. To minimize disk I/O, leave the batch logging parameter set to "ON".

Tuning Security

Authentication Server Modes

The two components of the Authentication Server (that is, the broker and the providers) can run in either of two modes: inmemory mode or ORB mode. The functionality is the same for either mode; the differences are in performance and opportunities for distribution. For more information, see "Authentication and Restriction" in the *Security Guide*.

Tuning Operating System and Network

TCP Tuning

Because the HTTP protocol runs on top of TCP, you can significantly improve performance by tuning some key TCP parameters.

The following table lists TCP parameters introduced with Solaris 2.6 and their recommended values:

Table 4–2 TCP parameters (Solaris only)

Parameter	Default	Recommended Value	Description
tcp_conn_req_max_q	128	10240	Maximum queued connections with a complete handshake. A high value prevents tcpListenDrops in case of a high request rate. This must be less than or equal to tcp_conn_re_max_q0.
tcp_conn_req_max_q0	1024	10240	Maximum queued connections with a handshake incomplete.
tcp_slow_start_initial	1	2	Addresses slow start bug in Windows and BSD TCP/IP stacks.
tcp_xmit_hiwat	8192	32768	TCP transfer window.
tcp_recv_hiwat	8192	32768	TCP receive window.

For more information about these parameters, see "Harnessing the Benefits of Solaris 2.6" on page A-3.

The following table lists additional parameters you can adjust based on the number of users and their locations relative to the server.

Table 4–3 Additional TCP parameters (Solaris only)

Parameter	Description
`tcp_close wait interval`	This is the number of microseconds an inactive connection will be "remembered". The shorter this number is, the smaller the connection table must be searched to identify a connection. If most users are local, then 60000ms may be sufficient. If the user population is very wide spread, then 240000 may be required as is recommended in RFC 1122.
`tcp_conn_hash_size`	This is the kernel hash table size for managing active TCP connections. A larger value makes searches far more efficient when there are many open connections. On Solaris, this value is a power of two and can be set as small as 256 (default) or as large as 262144 as is typically used in benchmarks. A larger tcp_conn_hash_size requires more memory, but it is clearly worth the extra investment if many concurrent connections are expected.

To set the TCP parameters described in Table 4–2 and Table 4–3, use these commands:

```
$ su root
# /usr/sbin/ndd -set /dev/tcp parameter value
```

where *parameter* is a parameter listed in the first column of the table above, and *value* is a value listed in the second column. The system startup files must be edited to make the changes survive a reboot.

Note: The `tcp_conn_hash_size` parameter must be adjusted in the /etc/system file by adding the line (modified to contain the desired hash table size, of course). For example:

```
set tcp: tcp_conn_hash_size=32768
```

You must then reboot the system for the hash size parameter to take effect.

For more information about these parameters and other TCP tuning for improving performance, see Adrian Cockcroft's *Sun Performance and Tuning*, 2nd Edition, 1998. It is essential to adjust the TCP parameters appropriately because they have a 20% to 30% impact on listener performance.

5

Monitoring Performance Statistics

This chapter discusses two performance monitoring tools for Oracle Application Server. The **oasomo** utility allows you to monitor your entire site, from the process-level to overall CPU usage. The **flexmon** utility allows you to monitor performance statistics from the command line.

Contents

The oasomo Utility

Overview

The **oasomo** utility is a dynamic monitoring utility that tracks system resources and monitors your site.

For a list of terms used in this chapter, see "Terminology" on page 5-32.

How does it work?

Oracle Application Server uses a C/Java API to publish an extensive list of metrics, or performance statistics, and uses the Oracle Application Server ORB to export metrics to the **oasomo** utility.

You can specify which metrics you want to monitor and configure **oasomo** to display them in tables and charts.

The **oasomo** utility also includes an EJB/ECO Performance Display tool that allows you to monitor EJB and ECO applications with greater detail than the basic **oasomo** window allows.

Running oasomo

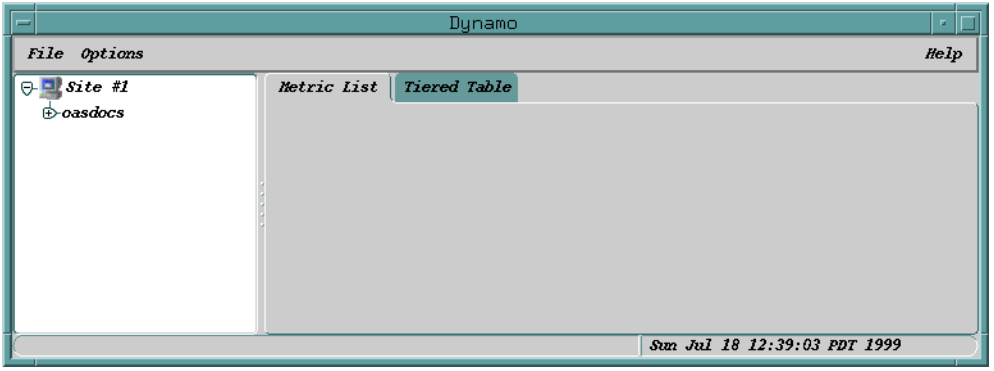
To start **oasomo**, enter the command:

```
% oasomo
```

The executable is in the **$ORACLE_HOME/orb/4.0/bin** directory for Unix installations and in **$ORACLE_HOME/orb/bin** for Windows NT installations.

Figure 5–1 shows the **oasomo** screen at startup.

Figure 5–1 oasomo's starting screen



oasomo Walk Through

The following steps offer a quick tour of monitoring an application with **oasomo**. This example assumes that you have an application added to your site.

The application in this example is a JServlet application named **hello**. It has a cartridge named **helloCart**.

For a list of terms used in this chapter, see "Terminology" on page 5-32.

> **Note:** This walk through only demonstrates viewing metrics for an application, it does not explain or describe values that these metrics should have during normal operation or what values could be an indication of performance problems.

1. Start the **oasomo** utility.

   See "Running oasomo" on page 5-2 for more information on running **oasomo**.

2. Start your application.

   See the Developer's Guide for your application for information on invoking it.

3. In the oasomo window, expand the *hostname* branch of the directory tree on the left. In Figure 5–1, this is the oasdocs branch. This will display your host's Nouns. Your application's name and PID is in this list.

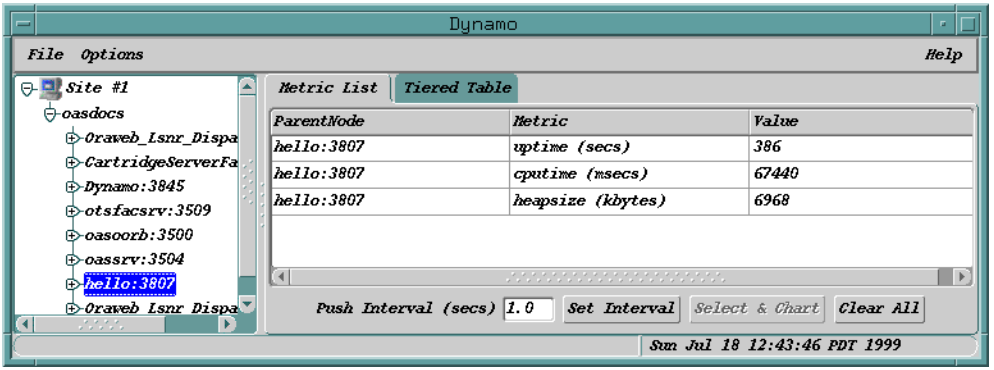
   A *Noun* is a component of an Oracle Application Server site. The host's Nouns in this example are Oracle Application Server processes.

4. Select your application from this list. This will bring up three of the metrics that are available for your application: uptime, cputime, and heapsize. Descriptions of these values can be found in Table 5–2, "Predefined process level metrics," on page 5-11.

   A *metric* is a statistic that is collected for a Noun.

   Alternatively, you could expand the application branch and select each of these metrics individually.

Figure 5–2 Application metrics in oasomo



The figure shows that for the hello application:

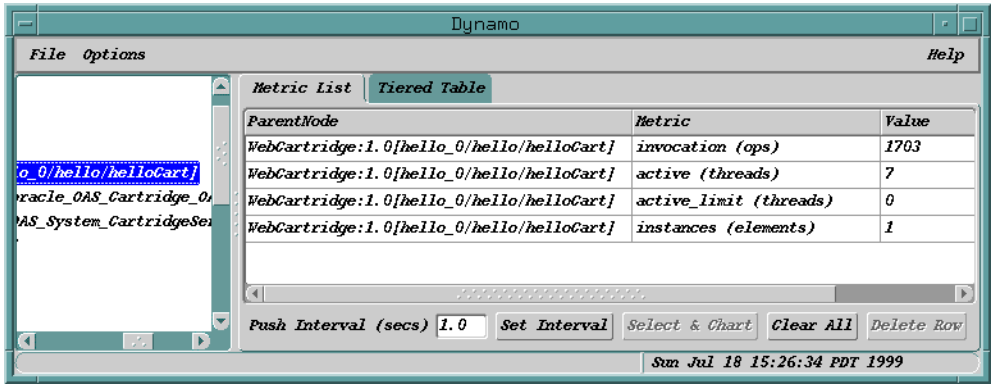
- 386 seconds have passed since the process was created
- 67440 milliseconds of CPU time have been used
- 6968 kilobytes of heap space have been allocated.

5. To view cartridge metrics for your application, expand the application branch. Next expand the CORBA_Objects branch. Select the element that ends with your cartridge's name. You may need to scroll the directory tree to the right to see this.

   Since our cartridge name is helloCart, we select the **IDL:oracle/OAS/Cartridge/Web/WebCartridge:1.0[hello/helloCart]** entry. This is shown in Figure 5–3.

   The cartridge metrics given are invocation, active, active_limit and instances. The descriptions can be found in Table 5–3, "Predefined CORBA interface metrics," on page 5-12.

Figure 5–3 Cartridge metrics in oasomo



The figure shows that:

- helloCart's methods have been invoked 1703 times
- there are currently 7 invocations of helloCarts's methods
- there is no limit on the number of concurrent invocations for helloCart methods (indicated with a 0)
- there is currently one active instance of helloCart.

6. You can also view queuing metrics for your application. Under your application's name in the directory tree, select the ORB_Queues branch. This will display the queuing metrics for this application.

   The queuing metrics given are qlen, maxMesgs, lastServed, totalWait, totalEnqueued, maxWait, numClasses and numQueues. The descriptions can be found in Table 5–7, "Predefined ORB queue metrics," on page 5-14.

Figure 5–4 Application queuing metrics in oasomo

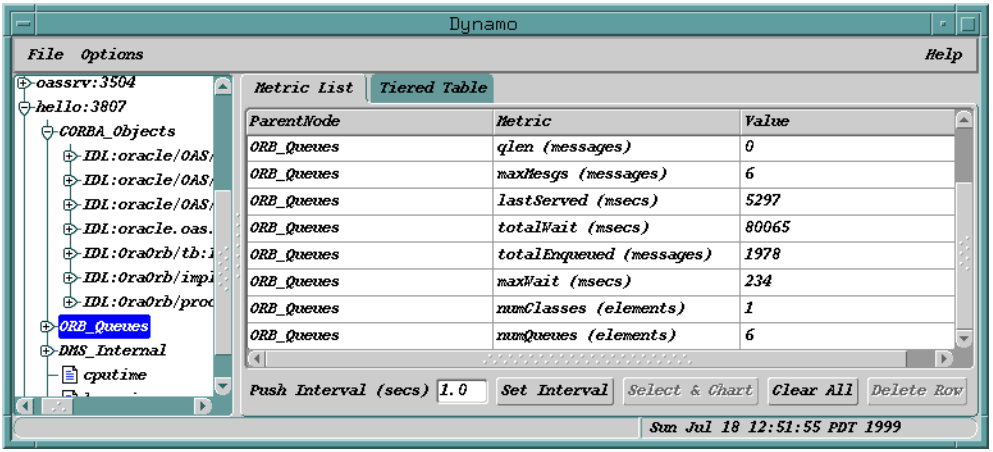


Figure 5–4 shows the following about this process of the hello application:

- the total queue length is 0
- there have not been more than 6 messages in any one queue at any time
- it has been 5297 milliseconds since a message has been serviced from the queue
- messages have spent a total of 80065 milliseconds waiting in the queue
- 1978 messages have been queued
- no message has waited for more than 234 milliseconds to be serviced from the queue
- there is only 1 queue class
- there are 6 queues.

Parts of the oasomo Window

This section discusses the different parts of the oasomo main window:

- oasomo Menu Bar
- Directory Tree
- Status Message Area
- ToolTips

oasomo Menu Bar

The Menu Bar provides the File, Options and Help menus.

The File menu presents three options:

- View EJB/ECO Metrics — to start the EJB/ECO Performance Display tool. For information, see "Viewing EJB and ECO Metrics" on page 5-17
- Print — to print the window
- Exit — to exit **oasomo** and close the window.

The Options menu presents three checkboxes:

- Show Data as Rate — displays the integer metrics in tables as a rate (value/second). **oasomo** calculates the rate based on an average over the previous five seconds. If no new data was received for a metric within the past five seconds, its entry will be left blank in the table. With this option selected **oasomo** displays a "-" for floating point and string values.
- Display Long Names — By default, **oasomo** displays the shortest possible name in the ParentNode column. Selecting the Display Long Names checkbox expands the ParentNode names as full path names for the path in the **oasomo** directory tree.
- Directory Tree — When this checkbox option is deselected, **oasomo** fills the entire display with the Metric List or Tiered Table data (and hides the directory tree).

The Help menu provides options for displaying online help. The menu presents choices for displaying the online Help contents, the index, and the search dialog. There is also an option for hiding any visible help windows.

Directory Tree

In the left column, the directory tree displays a hierarchical directory of all currently available Nouns and metrics for the current site. This list is dynamic, so that **oasomo** updates the directory list as programs start, and removes entries when programs stop.

When oasomo starts, only the hostname is shown. Expand its branch by clicking on the plus sign (+) next to its name. This will display all of the Process Nouns that are available for monitoring.

Selecting a Noun starts a subscription for all the Noun's metrics but not the metrics for any sub-Nouns. To start a subscription for a Noun or metric, select it. Once a

subscription is active, **oasomo** begins displaying metrics in the Metric List or Tiered Table.

Status Message Area

The bottom text area displays status messages. Note that the time status on the right side of the status area does not display the current system time. The time shown is the last update time for any **oasomo** display (hidden or visible). If data is not being collected, or the push interval is long, the clock updates infrequently, or not at all.

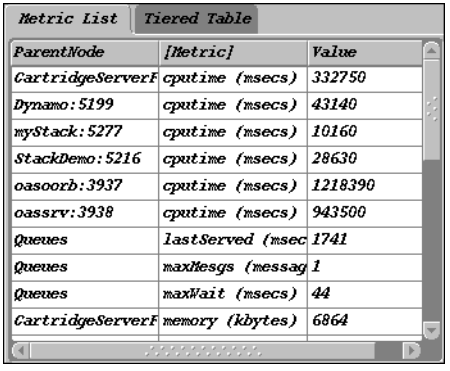
ToolTips

oasomo provides ToolTips for items in the Metric List and the Tiered Table. When the mouse moves over an item, and help is available, **oasomo** provides descriptive text about the item.

Metric List

Figure 5–5 shows a sorted Metric List with values supplied for each metric:

Figure 5–5 Sorted Metric list



The Metric List form contains three columns and up to 100 rows:

Column	Description
ParentNode	Displays the metric's parent Noun.
Metric	The statistic being gathered and its measurement unit.
Value	The current metric value.

This section describes the tasks available using the Metric List form:

Table 5–1 Metric list form

Action	Description
Adding Metrics	Click on a Noun or metric in the directory tree to add items to the Metric List. If you select a Noun, then **oasomo** adds all the Noun's metrics to the Metric List.
Removing Metrics	Remove metrics by clicking the Clear All button or by selecting one or more rows in the Metric List, and then clicking the Delete Row button. If metrics are not available, **oasomo** disables the Delete Row and Clear All buttons.
Sorting Metrics	**oasomo** displays path names in the ParentNode column, metric names in the Metric column and metric values in the Value column. Enable column sorting by clicking on a column heading. Sort mode sorts columns in descending order. When you enable sorting, the label for the column will be hilighted. See Figure 5–5 for an example of a sorted Metric List. **Note:** If you add metrics to a sorted list, they will be added to the bottom of the list. They will not be sorted.
Setting the Push Rate	**oasomo** displays the Set Interval button and the Push Interval value in the control area at the bottom of the Metric List or the Tiered Table. The push interval shows the time in seconds for the data update interval and the sample interval. Setting this value lower tells **oasomo** to update the displayed values more frequently. update `rate = 1 / interval` Setting the push interval higher will improve performance. **oasomo** rejects Push Interval values if resources are limited, the supplied value is small, or if you enter an invalid interval (invalid intervals include negative values and very large or very small values).

Tiered Table

A Tiered table allows you to see more than one level, or tier, of the directory at one time. The Tiered Table shows a selected Noun and its metric values as well as its children and their metric values. A Tiered Table contains a top row showing the selected Noun's metrics. The bottom rows show the metrics for Nouns contained within a selected Noun. When a Noun contains only Nouns, the top row in the Tiered Table displays blank values.

Figure 5–6 oasomo tiered table

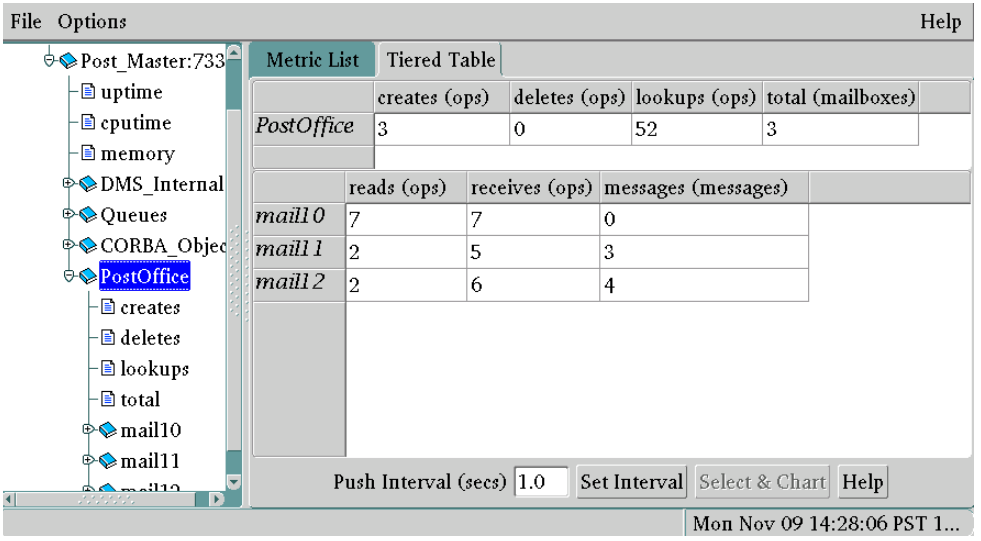


Figure 5–6 shows a Tiered Table display for a Post Office demonstration program. The top row shows the Post Office metrics: creates, deletes, lookups, and total mailboxes. The bottom rows show the Nouns, or in this example the individual post box Nouns within the Post Office, including: mail10, mail11, and mail12.

Columns in the bottom rows of the Tiered Table present metric values for each unique metric. Figure 5–6 shows values for the following metrics: reads, receives, and messages. In a Tiered Table, blank fields are possible for rows that do not contain a metric that is common across all Nouns.

To clear or remove values from a Tiered Table use the directory tree to select a new Noun. The Tiered table updates to display values for the currently selected Noun.

When new metrics are added to a Noun, due to new processes being added, changes in the state of the program containing the instrumentation, or for other reasons, **oasomo** updates the table to reflect the changes. If a row does not contain valid metrics because a subscription is no longer active, **oasomo** grays out the row.

oasomo supports sorting for the columns in the bottom rows of the Tiered Table. Select sort mode for a given column by clicking on the column heading. **oasomo** sorts rows in descending order and highlights the sorted column header.

Once a Tiered Table subscription is active, you can drill up the Metric Directory hierarchy by selecting the row label from the top row of the Tiered Table, or you can drill down by selecting a row label from any bottom row of the Tiered Table.

Metrics Available for Monitoring

The following tables show the metrics that can be monitored for Nouns. To view these metrics, either click on the described Noun or expand the Noun's branch on the directory tree and select the metric directly.

Under the *hostname* branch, the process Nouns are visible. Each process Noun contains these metrics:

Table 5–2 Predefined process level metrics

Metric Name	Description
uptime	The time, in seconds, since start of process.
cputime	The amount of CPU time, in CPU milliseconds, used by process.
heapsize	The amount of heap space allocated by the process in kilobytes.

Each process when expanded in the directory tree, displays the metrics shown in Table 5–2 and some or all of the following Nouns:

- CORBA_Objects
- DMS_Internal
- Dispatcher (only for Listener processes)
- Host_Metrics (only for Cartridge Server Factory process)
- ORB_Queues

CORBA_Objects

Table 5–3 shows the predefined metrics which are collected for every CORBA interface implementation. CORBA interface implementations are the basic building blocks for most of the Oracle Application Server components, such as EJBs, servlets, PL/SQL cartridges and internal components.

These metrics are available through this path: *hostname* > *process_noun* > CORBA_Objects > *interface*.

Table 5–3 Predefined CORBA interface metrics

Metric Name	Description
invocation	The number of times that any of this implementation's methods were invoked.
active	The number of currently executing invocations of any of this implementation's methods.
active_limit	The maximum allowed number of concurrently executing invocations of any of this implementations methods. A value of 0 means that no limit is set. A value of -1 means that the limit is infinite.
instances	The number of live objects for this CORBA implementation.

Table 5–4 shows the predefined CORBA_Object method metrics. If a method is never invoked, then **oasomo** does not list the method or its metrics. These metrics are available through this path: *hostname* > *process_noun* > CORBA_Objects > *interface* > *method*.

Table 5–4 Predefined CORBA method metrics

Metric Name	Description
invocation	The number of times that this method was invoked.
invocation_time	The total time elapsed, in milliseconds, while this method was invoked.
waitTime	The total time elapsed, in milliseconds, while requests for this method waited in ORB queues.

DMS_Internal

The **oasomo** utility is able to monitor the DMS spy in each process, and displays its performance statistics here. Within DMS_Internal, there are two Nouns: Measurement and Spy.

Table 5–5 shows the predefined internal process metrics. These metrics are available through this path: *hostname* > *process_noun* > DMS_Internal > Spy.

Table 5–5 DMS_Internal spy metrics

Metric Name	Description
pubPush	The number of times that process sent metric data to clients.
subscribeAttempt	The number of times that a client attempted to subscribe.
subscribe	The number of times that a client successfully subscribed.
cancel	The number of times that a subscription was canceled.
pubListeners	The number of clients listening for metric directory updates.
subscriptions	The number of active subscriptions.
resourcesUsed	The number of metrics per second subscribed by all clients of this spy.

Table 5–6 shows the predefined internal Measurement metrics. These metrics are available through this path: *hostname* > *process_noun* > DMS_Internal > Measurement

Table 5–6 DMS_Internal measurement metrics

Metric Name	Description
eventCreate	The number of times that an event was created.
nounCreate	The number of times that a Noun was created.
nounDestroy	The number of times that a Noun was destroyed.
stateCreate	The number of times that a state was created.
stateDestroy	The number of times that a state variable was destroyed.
eventDestroy	The number of times that an event was destroyed.
sampleVal	The number of times that the spy sampled any metric value.
getPub	The number of times that a client requested this process' metric list.
lastID	The most recent publication ID allocated.

Dispatcher

(only under the Listener processes) The Dispatcher section contains metrics for cartridges in your site. For each cartridge, such as *app_name* / *cart_name*, **oasomo** provides these metrics:

- TotalRequestWaitTime — The cumulative time spent waiting in the dispatcher for all requests for this cartridge type.
- TotalRequests — The total number of requests for this cartridge type.

Host_Metrics

(only under the Cartridge Server Factory process) Within Host_Metrics, you can subscribe to CPU or Virtual Memory metrics. The host metrics vary depending on your platform.

ORB_Queues

The **oasomo** utility monitors several queue level metrics for each Oracle Application Server process.

Table 5–7 shows the predefined queue metrics which are automatically gathered for each process. These metrics are available through this path: *hostname* > *process_noun* > ORB_Queues.

Table 5–7 Predefined ORB queue metrics

Metric Name	Description
qlen	The current total queue length.
maxMesgs	The maximum number of messages contained in any one queue at any one time.
lastServed	The time elapsed, in milliseconds, since a message was serviced from any queue.
totalWait	The total amount of time, in milliseconds, that all messages have spent waiting in queues.
totalEnqueued	The total number of messages that have been queued.
maxWait	The maximum amount of time (in milliseconds) that any one message spent waiting in any queue.
numClasses	The number of queue classes.
numQueues	The number of queues.

The Queues section also displays predefined metrics which are automatically gathered for each queue. These metrics are available through this path: *hostname* > *process_noun* > ORB_Queues > *queue*.

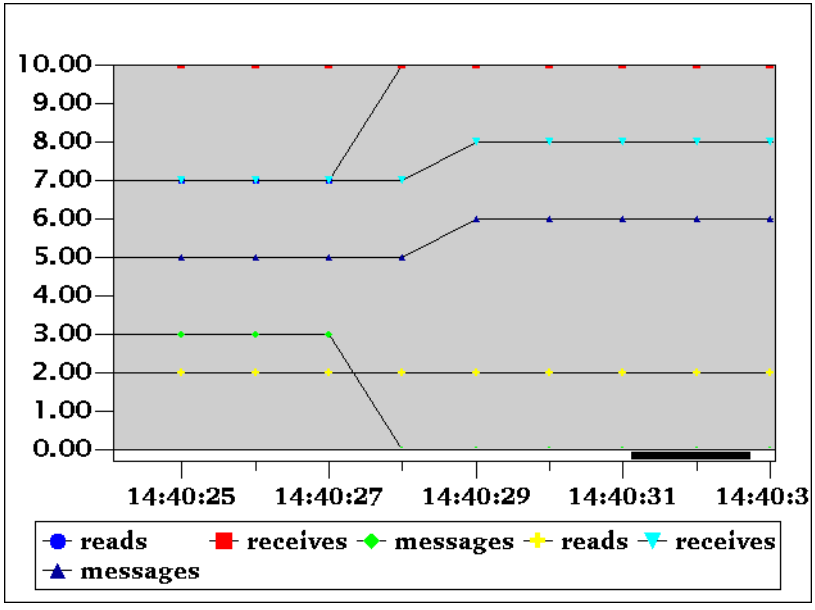
Table 5–8 Predefined queue metrics

Metric Name	Description
maxLength	The maximum number of messages contained in the queue at any one time.
qlen	The current length of the queue.
totalQueued	The total number of messages that have been sent to the queue.

Displaying Charts

A StripChart graphically displays the value of metrics over time. Figure 5–7 shows a sample StripChart.

Figure 5–7 oasomo StripChart



To bring up a Strip Chart, first select the metric or metrics you want to display from either the Tiered Table or the Metric list, and then push the Select & Chart button. If no metrics are selected, **oasomo** disables the Select & Chart button.

As **oasomo** receives data, at the rate specified by the Push Interval, it adds the new data to the StripChart. Older data moves to the left and off the display; this data is available by selecting and moving the scrollbar at the bottom of the chart (the black bar shown in Figure 5–7).

The StripChart displays update times on the x-axis and metric value ranges on the y-axis. The legend, found below the x-axis, displays names and symbols for the StripChart's metrics. Holding the cursor over a metric name in the legend displays the metric's full path name in the status area at the bottom of the StripChart.

Zooming In and Out

You can zoom in by pressing and holding the Shift button while left-clicking the mouse. You can zoom out by right-clicking the mouse.

StripChart Menu Bar

The StripChart Menu Bar provides the File and View menus. The File menu presents three selections:

- Clear Data
- Cancel Subscription to retain old data and cancel the subscription, disabling updates
- Close Graph to close the StripChart and close the window

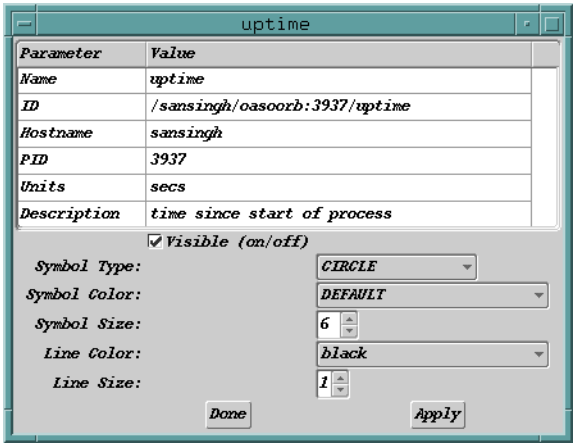
The View menu presents two options:

- Show Data as Rate
- Show Legend

Displaying the StripChart Legend Dialog

Clicking on a metric name in the legend displays the metric legend dialog:

Figure 5–8 oasomo StripChart legend dialog



For each metric, the legend dialog allows you to alter any of the following: the symbol shape, the symbol color, the symbol size, the line color, and the line size. A metric visibility checkbox allows you to disable a metric's StripChart display. When a metric is not visible, its name is still available in the legend so that you can bring up the legend dialog to make the metric visible again.

Viewing EJB and ECO Metrics

This section describes the EJB/ECO Performance Display, the dynamic performance monitoring utility for Enterprise Java Beans (EJB) and Enterprise CORBA Objects (ECO). It is intended for use by Oracle Application Server administrators and EJB developers.

> **Note:** This tool supports both EJBs and ECOs. In fact, it cannot distinguish between EJBs and ECOs. Therefore, the remainder of the document only refers to EJBs.

This utility consists of a set of screens that present performance metrics and system organization for an Oracle Application Server site executing EJBs. The screens provide an administrator or developer easy access to performance data about EJB applications and their components.

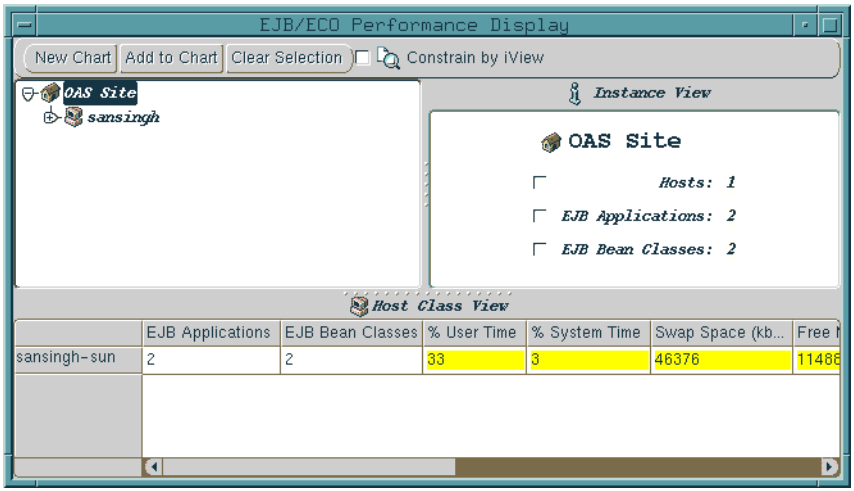
Note: For a definition of terms such as metric or Noun, see "Terminology" on page 5-32.

Throughout this section, the following terminology is used:

- **iView** — Instance View
- **cView** — Class View

Figure 5–9 shows a sample EJB/ECO Performance Display window.

Figure 5–9 The viewer's starting screen



The window has three main views, the directory tree on the left, the Instance view (iView) on the right, and the Class view (cView) on the bottom. This section describes these views.

Directory Tree

In the left column of the window is an expandable, scrollable tree control that presents the entire tree of Nouns and shows their containment relationships. Each Noun type is shown alongside a graphical icon that indicates its type.

At startup, the directory tree shows the single Site and all of the site's Host Nouns. Deeper levels of the tree may be explored by clicking the plus (+) symbol. If the tree

is too large to fit within the directory tree frame then scroll bars appear to allow you to scroll horizontally or vertically.

When any node is selected (with a single mouse click) in the directory tree view, the corresponding Noun displays in the iView.

Double clicking on a node not only selects the instance, but also expands the node in the tree (when possible).

Nodes in the directory tree may be selected with explicit mouse clicks on the icons within the window or when you make selections in other views. For example, selecting a row in the cView or selecting a metric in a Chart causes the corresponding Noun to be selected in the directory tree.

EJBs in the directory tree are not strictly grouped according to their Java class hierarchy but instead form a single group beneath the application node. In other words the organization under an EJB application is a flat collection of Bean classes even though the classes themselves may form a rich Java hierarchy.

The directory tree view is updated automatically whenever instances in the system are created or destroyed. Oracle Application Server buffers rapid updates in the underlying system for performance reasons, so the directory tree will update on regular intervals (approximately 10 seconds).

Instance View (iView)

The Instance view displays all the performance metrics that are available for a particular Noun. Different Noun classes have different instance views (e.g., a Host Noun's metrics are different from Method Noun's metrics).

The Instance view's data values are updated automatically every three seconds. If a metric value has not changed in more than 30 seconds, then the value is painted on a white background. If the metric value has changed more recently than 30 seconds, then it is painted on a yellow background. The intensity of the highlighted background diminishes as the data becomes older.

oasomo displays each metric in an iView with a checkbox. If you wish to create a new chart that includes the metric or add the metric to an existing chart, select the metric's checkbox and choose the appropriate button from the toolbar.

You may select the Noun to be displayed in the iView by selecting the Noun in the directory tree, by selecting a cView row corresponding to the Noun, or by selecting any of the Noun's metrics in a Chart. Whenever you select a Noun for the iView, its directory tree node is also selected in the directory tree and the directory tree's branch is expanded, if necessary.

When a new Noun is selected for the iView, the EJB/ECO Performance Display clears all iView checkboxes and forgets which metrics were chosen.

At startup, the iView displays the single Site Noun.

Class View (cView)

The cView allows you to view all instances of a particular metric class simultaneously. Each instance is represented with a single row in a scrollable table. Columns of the table contain metric values for each of the metrics listed for the chosen class. Only one class is visible at a time. The view's rows may be sorted by any of its columns by clicking on the corresponding column header. You may select any class from the list of available classes.

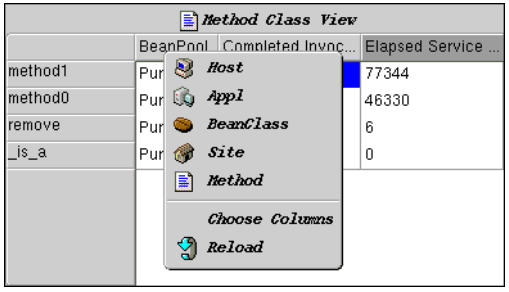
If you select the “Constrain by iView” checkbox at the top of the window, then selecting a node in the directory tree may change the cView because the new selection may constrain the cView differently than the previous selection did.

At startup, the cView displays the Host Noun class and all sorting is turned off. To select a different class, you select the class from the cView popup menu. To use the cView popup menu, place the mouse pointer anywhere in the cView and right-click the mouse button. A menu then appears that allows you to:

- pick a new Noun class
- choose the columns to be displayed in the cView
- refresh the cView.

Figure 5–10 shows the cView popup:

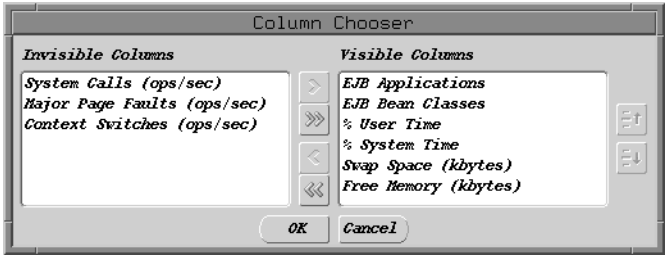
Figure 5–10 the cView popup menu



If the cView is too short to fit the entire cView popup menu then the menu includes a vertical scrollbar. If the cView is too narrow to fit the cView popup, then the tool does not display the menu.

To choose which columns to display in the cView, select Choose Columns from the cView popup menu. The column selection window as shown in Figure 5–11 will appear.

Figure 5–11 Class View column selection window



To select and order columns use the shuttle controls to move column headers back and forth between the left-hand column of the shuttle and the right. Headers that appear in the left column of the shuttle will be made invisible in the cView, those on the right will be visible in the order in which they appear in the shuttle control. Press the OK button to accept the choices or Cancel to leave the cView unchanged.

The EJB/ECO Performance Display remembers the current column choices for each of the Noun classes, so if you select a new Noun class for the cView and then return to the previously displayed class, then the previously selected columns will reappear.

The cView's cells display the metric values. The EJB/ECO display highlights cView metrics in the same way as iView metrics and updates the data values every three seconds. If a metric value has not changed in more than 30 seconds, then the value is displayed on a white background. If the metric value has changed more recently than 30 seconds, then the data displays with a yellow background. The highlighted background fades as the data ages.

You may select any cell in the cView table by clicking on it. To select multiple cells or to de-select cells, press the CTRL key while clicking. You can also select a region of cells by clicking on one of the cells and dragging the mouse while keeping the mouse button depressed. To select an entire row of cells, click on the row's header. Clicking on a row header causes the corresponding Noun to be selected in both the directory tree and the iView. To clear all selections, press the Clear Selection button.

By default, the rows of the cView are not sorted. To sort by a particular column, click on the column's header. When the cView is sorted, the tool highlights the corresponding column header. Sort order is always descending.

As new Nouns are dynamically discovered and added to the directory tree, they are not automatically added to the cView, even if it is currently displaying a new Noun's class. To force the cView to display new Nouns, select Reload from the cView popup menu.

Buttons and Checkboxes

This section describes other options in the EJB/ECO display.

New Chart Button The New Chart button activates whenever you select one or more metrics in either the iView or the cView. When you press New Chart, the EJB/ECO display launches a new Chart view and populates it with the selected metrics. To make a new chart, you may select metrics from the iView, the cView or both.

Add to Chart Button The Add to Chart button activates if you have selected metrics while one or more Charts are already active. Selecting Add to Chart adds the selected metric to the existing chart. If there are more than one charts active, then a dialog pops up that allows you to select the Chart to receive the new metrics.

For more information about charts, see "Displaying Charts" on page 5-23.

Clear Selection Button The Clear Selection button activates whenever you have selected metrics. When you select Clear Selection, the tool de-selects all selected metrics.

Constrain by iView Checkbox The Constrain by iView Checkbox allows you to limit or filter the potentially large number of Nouns displayed in the cView. When the Constrain by iView checkbox is selected, the Nouns in the cView are limited (constrained) to only those that are descendants of the Noun that is currently selected in the iView.

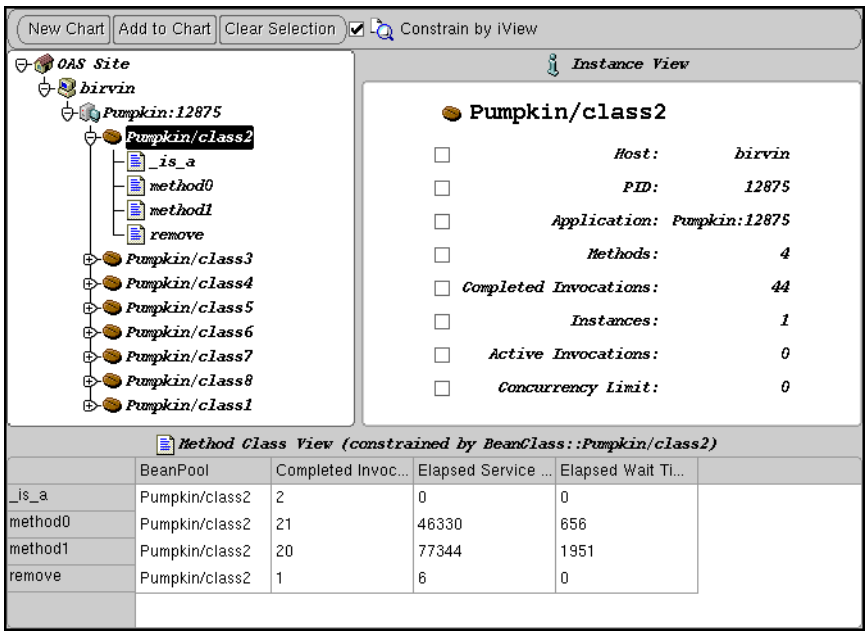
For example, if you are only interested in the methods for a particular EJB application, then you can limit the cView to the application with the Constrain Checkbox. To do this, display the desired application in the iView and select the Constrain by iView Checkbox. Checking the box instructs the EJB/ECO display to fill the cView with only those Nouns that descend from the selected application. Finally, select "Method" in the cView popup menu.

Sometimes, constraining produces an empty cView. For example, if you select the Host class in the cView and then constrain the cView with a Method Noun, then the cView will be empty because methods are never ancestors of hosts.

Figure 5–12 shows an example use of the Constrain by iView checkbox. The user has selected the Method class in the cView and constrained the cView by the Bean Class named Pumpkin/class2. The result is a cView that shows only those methods contained by the Pumpkin/class2 Bean class.

When the user de-selects the Constrain by iView checkbox, the tool immediately removes constraints from the cView and the cView displays all Nouns of the selected Noun class.

Figure 5–12 Class View constrained by an instance view

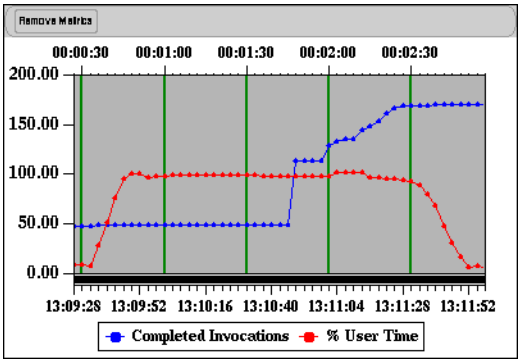


Displaying Charts

You can create charts by selecting the New Chart button in the EJB/ECO display tool bar. Charts graph one or more metrics over a moving time axis. The EJB/ECO display updates all charts every three seconds.

Figure 5–13 shows an example chart with two metrics. The bottom of the chart contains a legend that displays the names of the chart's metrics alongside icons that match the metric names to the curves in the chart display area. To see detailed information about the charted Nouns click on a metric in the legend. The EJB/ECO display will then display the corresponding Noun in the iView and select the Noun in the directory tree.

Figure 5–13 EJB/ECO display chart with two metrics



The t-axis (bottom axis) shows the current time of day. The y-axis (left-hand axis) shows the range of metric values for the chart. Charts always use one y-axis and all curves are mapped to it. This sometimes creates charts in which one or metrics distort the y-axis so that some metrics are not readable in the chart. In these situations, you may create a new chart to separate the metrics. The top of the chart is labeled with a t-axis that starts at 0 and increments every 30 seconds. The EJB/ECO display draws a vertical bar through the chart at 30 second intervals.

Charts contain one control button, the Remove Metrics button. When you select the Remove Metrics button, the EJB/ECO display shows a two column shuttle window. Use the shuttle controls to move metric names to the left or right column. When you press the shuttle's OK button, the tool deletes all metrics in the left column of the shuttle and keeps all metrics listed in the right column. Deleted metrics may not be re-displayed through the Remove Metrics button, but they may be replaced using the Add to Chart button in the main window.

Zooming In and Out

You can zoom in by pressing and holding the Shift button while left-clicking the mouse. You can zoom out by right-clicking the mouse.

Noun Classes

Table 5–9 displays the available metrics for each Noun class:

For a list of terms used in this chapter, see "Terminology" on page 5-32.

Table 5–9 EJB/ECO Performance Display Noun classes and their metrics

Noun Class	Metric	Available Metrics
Site	Hosts	Number of hosts running EJBs.
	EJB Applications	Total number of EJB applications on all hosts.
	EJB Bean Classes	Total number of Bean Classes contained in all applications on all hosts.
Host	EJB Applications	Number of EJB applications running on the host.
	EJB Bean Classes	Number of Bean Classes contained in all applications on the host.
	% User Time	Current user CPU utilization.
	% System Time	Current system CPU utilization.
	Swap Space	Available swap space.
	Free Memory	Available physical memory.
	System Calls	Current System call rate.
	Major Page Faults	Major Page Fault rate.
	Context Switches	Thread context switch rate.
Application	Host	The name of the host machine running this application.
	PID	The process ID number.
	EJB Bean Classes	Total number of EJB Bean Classes contained within the application.
	% CPU	Current CPU utilization.
	Heap Size	Heap space used by the application process.
	ORB Enqueues	Current ORB enqueue rate.
	Uptime	Application uptime.
	ORB Queue Length	Current ORB queue length.

Table 5–9 EJB/ECO Performance Display Noun classes and their metrics

Noun Class	Metric	Available Metrics
Bean Class	Host	The name of the host machine running this application.
	PID	The process ID number.
	Application	The name and PID of the application.
	Methods	Number of methods implemented by class
	Completed Invocations	Number of method invocations completed
	Instances	Number of currently active instances of the class
	Active Invocations	Number of method invocations currently active
	Concurrency Limit	Maximum allowed concurrency level for the bean
Method	Host	The name of the host machine running this application.
	PID	The process ID number.
	Applications	The name and PID of the application.
	Bean Class	The bean's class name.
	Completed Invocations	Number of completed method invocations.
	Elapsed Service Time	Total elapsed time spent during invocations of the method.
	Elapsed Wait Time	Total time spent waiting in ORB queues by requests for the method.

Note: The update interval cannot be changed for the EJB/ECO display tool. The tool's update interval is three seconds.

The flexmon Utility

Overview

This utility can be used for quick and easy monitoring of Oracle Application Server. The **flexmon** utility is similar to the UNIX commands vmstat and iostat; all three accept options, report statistics, and print them out to standard output.

The **flexmon** utility:

- is text based
- can be used for automated testing
- provides access to any / all metrics that Oracle Application Server exports
- has a configurable push interval
- can be used to discover, list, and monitor metrics
- associates data values with metric names in the output. Names can be short or long.

Description of Command Line Interface

Before running **flexmon**, the environment must be properly set up. Therefore, the CLASSPATH must include:

- $ORACLE_HOME / orb / 4.0 / classes / yoj.jar (Unix only)
- $ORACLE_HOME / orb / 4.0 / classes / dms.jar (Unix only)
- $ORACLE_HOME / orb / classes / yoj.jar (Windows NT only)
- $ORACLE_HOME / orb / classes / dms.jar (Windows NT only).

A particular command line can request to list the metric directory or to actually monitor the metrics. If monitoring, then the command line contains a space-separated list of metrics. It is possible to specify a push-interval and / or count by preceding the subscription with a '-i' flag for interval and / or a '-c' flag to specify a count. The flags may be used together.

Since **flexmon** is a java based utility, it must be run with the java interpreter.

Syntax

```
java flexmon [-listmetrics] [-d delimiter] [-s | -t]
             [-i interval] [-c count] [subscriptions...] [-help]
```

Arguments

Argument	Description
-listmetrics	Displays a list of all metrics.
-d *delimiter*	Specifies a delimiter character to be used as an alternate to the slash (/) character used to separate subcomponents of a metric or Noun.
-s	Displays the shortened name for metrics.
-t	Display only metric values (terse display).
-i *interval*	Specifies the interval, in seconds, between display updates. If no interval is given, 1 is used.
-c *count*	Specifies the number of intervals to display. If no count is given, only a break (^C) will stop **flexmon**.
subscriptions...	A list of one or more Nouns and metrics, separated by spaces, to view. To list all of the metrics for a Noun, specify the Noun's name.
-help	Displays the syntax of the **flexmon** utility.

Usage Examples

Example 5–1 Listing all available metrics

This example demonstrates listing all available metrics. Note that only a part of the output is given here.

```
% java flexmon -listmetrics
/
/myNode/
/myNode/otsfacsrv:6558/
/myNode/otsfacsrv:6558/DMS_Internal/
/myNode/otsfacsrv:6558/DMS_Internal/Measurement/
...
/myNode/Oraweb_Lsnr_Dispatcher:6567/Dispatcher/Cartridges/HelloWorld/hwcart/
...
/myNode/Oraweb_Lsnr_Dispatcher:6567/uptime
/myNode/Oraweb_Lsnr_Dispatcher:6567/cputime
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize
```

Example 5–2 Displaying more than one metric

```
% java flexmon /myNode/Oraweb_Lsnr_Dispatcher:6567/cputime
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize
Thu Jul 15 10:09:04 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/cputime  16140 msecs
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3024 kbytes
Thu Jul 15 10:09:04 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/cputime  16140 msecs
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3088 kbytes
^C
```

Example 5–3 Displaying metrics in abbreviated formats

The `-s` argument can be used to display a shortened name (without the path information) for each metric. Since we refer to a Noun in the syntax, all metrics under the Noun are displayed.

```
% java flexmon -s /myNode/otsfacsrv:6558/
Thu Jul 15 10:13:32 PDT 1999
uptime 7903 secs
cputime 12820 msecs
heapsize 2464 kbytes
Thu Jul 15 10:13:33 PDT 1999
uptime 7903 secs
cputime 12820 msecs
heapsize 2464 kbytes
^C
```

The `-t` argument only displays the metric values. The metric names are not displayed.

```
% java flexmon -t /myNode/otsfacsrv:6558/
7964 13010 2480
7965 13060 2528
7966 13070 2536
^C
```

Example 5–4 Changing the update interval

This example displays the updated metric value every 10 seconds.

```
% java flexmon -i 10 /myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize
Thu Jul 15 10:19:25 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3088 kbytes
Thu Jul 15 10:19:35 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3152 kbytes
Thu Jul 15 10:19:45 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3152 kbytes
^C
```

Example 5–5 Displaying a fixed number of updates

This example only displays the metric 3 times. After the three updates, the program quits. Note that execution was not broken in this example with a break character.

```
% java flexmon -c 3 /myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize
Thu Jul 15 10:21:17 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3152 kbytes
Thu Jul 15 10:21:17 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3216 kbytes
Thu Jul 15 10:21:18 PDT 1999
/myNode/Oraweb_Lsnr_Dispatcher:6567/heapsize  3216 kbytes
```

Example 5–6 Displaying a metric without delimiters

```
% java flexmon
/myNode/Oraweb_Lsnr_Dispatcher:6567/Dispatcher/Cartridges/HelloWorld
/hwcart/
Unknown metric
/myNode/Oraweb_Lsnr_Dispatcher:6567/Dispatcher/Cartridges/HelloWorld/hwcart/
```

This example fails because the Noun 'HelloWorld/hwcart' embeds a slash in its name. This causes confusion between slashes that are delimiters and slashes that are part of a Noun names.

This is corrected, using a delimiter, in Example 5–7.

Example 5–7 Displaying a metric with delimiters

It is important to chose a unique delimiter that will not create confusion between the metric list or your operating system. Plus signs (+) and comas (,) are good choices in most cases.

```
% java flexmon -listmetrics -d +
...
+myNode+Oraweb_Lsnr_Dispatcher:6572+Dispatcher+Cartridges+HelloWorld/hwcart+
...

% java flexmon -d +
+myNode+Oraweb_Lsnr_Dispatcher:6572+Dispatcher+Cartridges+HelloWorld
/hwcart+
Thu Jul 15 09:34:57 PDT 1999
+myNode+Oraweb_Lsnr_Dispatcher:6572+Dispatcher+Cartridges+HelloWorld/hwcart+Tota
lRequestWaitTime  0 msecs
+myNode+Oraweb_Lsnr_Dispatcher:6572+Dispatcher+Cartridges+HelloWorld/hwcart+Tota
lRequests  0 Count
Thu Jul 15 09:34:57 PDT 1999
+myNode+Oraweb_Lsnr_Dispatcher:6572+Dispatcher+Cartridges+HelloWorld/hwcart+Tota
lRequestWaitTime  0 msecs
+myNode+Oraweb_Lsnr_Dispatcher:6572+Dispatcher+Cartridges+HelloWorld/hwcart+Tota
lRequests  0 Count
^C
```

Terminology

The utilities described in this chapter use the following terminology:

Term	Definition
Application Noun	An application noun is a single process that runs an EJB or ECO application. oasomo labels each application noun with the deployment name of the corresponding application.
Bean Class Noun	A Bean Class Noun corresponds to a single Java class hosted by a single EJB application. If several different applications contain the same Java class or if several different instances of a single application are running, then each occurrence of the Java class is represented by a different Bean Class Noun. However, if a single application contains multiple beans all of the same class, then they are all represented with the same Bean Class Noun. A Bean Class is named with the name of the EJB application followed by a '/' character followed by the base name (.i.e., not the full path name) of the Java class of the EJB. For example, the Bean Class Noun corresponding to the Stack class within the 4.0.8 example EJB application known as StackServer is named “StackServer/Stack”.
Chart	A performance metric display that shows metric values over time.
Directory	A hierarchical collection of Nouns and metrics that may be monitored in Oracle Application Server.
Host Noun	An individual host (node) within an Oracle Application Server site. Host Nouns contain metrics that may vary depending on the platform due to operating system differences.
Method Noun	A Method Noun corresponds to a single method within a particular EJB Noun. Metrics for methods include count of invocations, time elapsed during invocations of the method, and time spent waiting in ORB queues before invocations. Method metrics are summed over all invocations of all instances of the EJB. **oasomo’s** EJB/ECO Performance Display tool does not distinguish between instances (individual beans) within an EJB class when calculating method metrics.
Metric	A performance statistic collected for a Noun, such as uptime or queue length.
Metric List	A list of statistics available for collection.
Noun	A component of an Oracle Application Server site. Examples of a Noun are component, object, cartridge, class, bean, process, piece-of-code, computer, process, etc.

Term	Definition
Push Interval	The interval in seconds wherein an object (a Spy) maintains a publication of available statistics and sends the requested statistics to the subscriber.
Site Noun	An entire Oracle Application Server site (i.e. website40). There is always exactly one site Noun in the directory tree.
Spy	A CORBA object that collects data, accepts subscriptions from client monitors, samples data and pushes data. Each Oracle Application Server process contains a Spy.
Subscriber	Any program that collects performance statistics from a Spy. For example, the **flexmon** utility is a subscriber.
Subscription	A list of metrics requested by a subscriber.
Tiered Table	A Tiered table allows you to see more than one level, or tier, of the **oasomo** directory at one time. A Tiered Table shows a selected Noun and its metric values and its children and their metric values. A Tiered Table contains a top row showing the selected Noun's metrics. The bottom rows show the metrics for Nouns contained within the selected Noun.

A

Operating System Tuning

This chapter provides performance tips for your operating system and your hardware. It also includes a summary of the performance benefits of Solaris 2.6. For more information about tuning and performance for Solaris, see Adrian Cockcroft's *Sun Performance and Tuning*, 2nd Edition, 1998.

Contents

Monitoring Processor Use

To determine process utilization, you should gather CPU statistics. You should also monitor system scalability by adding users and increasing the system workload. Use the **sar** (System Activity Reporter) and **mpstat** utilities to monitor process use.

Using the sar Utility

You can use **sar** to sample cumulative activity counters in the operating system at specified intervals.

Report CPU Utilization

To determine process use, use the following **sar** command:

```
$ sar -u 5 5
```

This command will sample CPU usage five times, in five second intervals.

You will get a listing similar to this:

```
$ sar -u 5 5
SunOS dummy-sun 5.5.1 Generic_103640-03 sun4u    03/02/99

14:53:48    %usr    %sys    %wio   %idle
14:53:53       0       0       0     100
14:53:58       1       0       0      99
14:54:03       0       0       0     100
14:54:08       3       3       0      93
14:54:13       0       0       0     100

Average        1       1       0      98
```

The **sar** command (-u option) provides the following statistics:

Table 5–10 CPU statistics, as reported by the sar utility

CPU Statistics	Description
%usr	percentage that the processor is running in user time
%sys	percentage of processes running in system time
%wio	percentage the processor spends waiting on I/O requests
%idle	percentage that the processor is idle

Using the mpstat Utility

The **mpstat** utility is similar to the **sar** command in that the first argument to **mpstat** is the polling interval time in seconds. The second argument to **mpstat** is the number of iterations.

The **mpstat** command:

```
$ mpstat 1 3
```

reports three processor statistics in one second intervals. For example:

```
$ mpstat 1 3
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    1   0    0   268   64  148   11    0    0    0    33    3   5   0  92
  0    5   0    0   250   49  157   13    0    1    0   357    2   0   0  98
  0    0   0    0   247   47  134    8    0    0    0   326    0   0   0 100
```

The **mpstat** utility reports the statistics per processor, as shown in the following table:

Table 5–11 CPU statistics, as reported by the mpstat utility

Statistic	Description
CPU	processor ID
minf	number of minor faults
mjf	number of major faults
xcal	number of inter-processor cross calls
Intr	number of interrupts
ithr	number of interrupts as threads
csw	number of context switches
icsw	number of involuntary context switches
migr	number of thread migrations to another processor
smtx	number of spins for a mutex lock, which means the lock was not obtained on the first attempt
srw	number of spins on reader-writer lock, which means the lock was not obtained on the first attempt
syscl	number of system calls
usr	percentage that the processor spent in user time
sys	percentage that the processor spent in system time
wt	percentage that the processor spent in wait time (waiting on an event)
idl	percentage that the processor spent in idle time

Harnessing the Benefits of Solaris 2.6

Sun Microsystems updates Solaris operating system components regularly, such as the Transmission Control Protocol / Internet Protocol (TCP / IP) subsystem, that are heavily used by Oracle Application Server. Make sure you have installed the latest patches.

Also available from Sun Microsystems is the Solaris Internet Server Supplement, which is a set of add-on modules specially tailored for Solaris systems that host Web sites.

Oracle recommends using Solaris 2.6, which was a performance upgrade designed to improve network traffic. There were several optimizations made which improve network performance.

Processor Sets

You can create processor sets that bind processes to a set of processors, as opposed to a single processor. There are two types of processor sets, user-created processor sets, and system-created processor sets.

User-created User-created processor sets can be managed using the **psrset** command, or the **pset_create( )** system call. Processors that are assigned to user-created processor sets only service light weight processes (LWPs) that are bound to that particular processor set. However, system-created processor sets can service other LWPs. System-created processor sets do not always exist on a particular system.

System-created System-created processor sets are useful when certain processors communicate more efficiently together than they do with other processors. System-created processor sets cannot be modified, nor removed, but you can bind processes to them. Processor sets are used to improve CPU use by binding a certain set of processes, such as the Oracle listeners or cartridge processes, to processor sets. Processor sets help reduce processor contention between the listener processes and the cartridge processes.

Tuning I/O

Direct input/output (I/O) bypasses the UNIX file system cache and copies the file system-based file data directly into user space. Direct I/O on file systems is similar to raw devices. Solaris 2.6 allows direct I/O to be performed using the `directio()` system call. An application can use the `directio()` system call to perform direct I/O processing on a file. To control whether or not direct I/O to the file system is forced, use the `mount` command options: `noforcedirectio` and `forcedirectio`.

Use direct I/O to:

- improve large sequential I/O performance
- improve performance of large files during file transfers
- eliminate extra buffer copies and file system cache maintenance
- reduce CPU consumption

Monitoring TCP Statistics

Solaris 2.6 uses a new TCP/IP statistic, known as `tcpListenDrop`, which counts the number of times that a connection is dropped because of a full queue. Use the Solaris **netstat** utility (-s option) to report networking statistics, including `tcpListenDrop`.

Applications increase the size of the queue by specifying higher values for the backlog to the `listen()` call. Set the maximum backlog size by adjusting the value of the `ndd` on the `tcp_conn_req_max` parameter. See Table 5–12 for a list of TCP parameters and their recommended values.

In the case of an initial handshake, an incoming packet is sent with only the Synchronized Sequence Numbers (SYN) flag set. When a packet is sent, the server makes an entry in the listen queue, and then sends another packet to acknowledge the first packet. It also includes a SYN flag to reciprocate the synchronization of the sequence number in the opposite direction. The client then sends another packet to acknowledge the second SYN, and the server process is then scheduled to return from the `accept()` call, subsequently moving the connection from the listen queue to the active connection list.

If you send an initial SYN packet, but it is not acknowledged with a second SYN packet, it will eventually time out, and the server will discard the client from the listen queue. The listen queue becomes full when a server has several SYN packets that do not contain valid source addresses. This causes new connections to wait for old connections to be discarded from the queue.

Solaris 2.6 corrects this problem by using two separate queues, as opposed to one. In addition, there are two new TCP tunable parameters, `tcp_conn_req_max_q` and `tcp_conn_req_max_q0`, which specify the maximum number of completed connections that are waiting to return from an `accept()` call, and the maximum number of incomplete handshake connections, respectively.

Use the **netstat** utility (-s option) to monitor TCP statistics, and to determine connection drop activity, as well as the type of drops, for example:

```
tcpListenDrop=25642 tcpListenDropQ0=0

tcpHalfOpenDrop=0
```

The `tcpHalfOpenDrop` statistic is incriminated when an in-doubt connection is dropped. The default value for `tcp_conn_rq_max_q` is 128, and the default value for `tcp_conn_req_max_q0` is 1024. The default values are typically sufficient and should not require tuning. However, by examining the statistics with the **netstat** utility, you can determine if the parameters need to be adjusted.

TCP Parameters

The following table lists TCP parameters introduced with Solaris 2.6 and their recommended values:

Table 5–12 TCP parameters (Solaris only)

Parameter	Recommended Value
tcp_conn_req_max_q	1024
tcp_conn_req_max_q0	1024
tcp_close_wait_interval	60000
tcp_rexmit_interval_min	1500
tcp_xmit_hiwat	65536
tcp_xmit_lowat	24576
tcp_recv_hiwat	65536
tcp_conn_hash_size	256
tcp_slow_start_initial	2

To set these TCP parameters, use these commands:

```
$ su root
# /usr/sbin/ndd -set /dev/tcp parameter value
```

where *parameter* is a parameter listed in the first column of the table above, and *value* is a value listed in the second column.

For more information about these parameters and other TCP tuning for improving performance, see Adrian Cockcroft's *Sun Performance and Tuning*, 2nd Edition, 1998.

Improving Startup Latency

TCP implementations use a congestion window that limits the number of packets that can be sent before an acknowledgment. This is used to improve the startup latency and also helps avoid overloading the network. The TCP standard specifies that the initial congestion window should consist of one packet, then double upon each successive acknowledgment. This causes exponential growth and may not necessarily be ideal for HTTP servers, which typically send small batches of packets.

Solaris 2.6 provides a `tcp_slow_start_intital` parameter that can be used to double the congestion window from its default of 1 to 2. This improves transmission throughput of small batch sizes.

Contrary to Solaris, NT version 4 does not immediately acknowledge receipt of a packet upon connection-start, which results in an increase in the connection startup latency. NT version 4 does, however, immediately acknowledge if two packets are sent. The difference between the NT and the Solaris implementation causes performance discrepancies, or higher response times, when NT clients are used to connect to Solaris servers with a high-speed or LAN-based network. To correct this, set the congestion window on Solaris to 2, using `tcp_slow_start_initial=2`.

Reducing Connection Backlog

In Solaris 2.6, the `tcp_conn_hash_size` parameter can be set to help address connection backlog. During high connections rates, TCP data structure kernel lookups can be expensive (memory consuming) and can slow down the server. Increasing the size of the hash table improves lookup efficiency. The default for `tcp_conn_hash_size` is 256. This parameter must be a power of 2, and can bet set in the **`/etc/system`** kernel configuration file.

Monitoring Network Traffic

Use the **netstat** utility to monitor the overall network traffic, as well as the network traffic for a given interface using the (-k) option. Solaris 2.6 has several new counters that report byte count statistics. The following table lists the **netstat** utility counter names and descriptions:

Table 5–13 Monitoring network traffic using the netstat utility

Utility Counter Name	Description
tcpListenDrop	reports how many connections were dropped because of network traffic
`rbytes`	read byte count
`obytes`	output byte count
`multircv`	multicast receive
`multixmt`	transmit count
`brdcstrcv`	broadcast byte count
`brdcstxmt`	broadcast transmit count
`norcvbuf`	buffer allocation failure count
`noxmtbuf`	buffer allocation failure count

The following table lists additional Solaris 2.6 network enhancements that increase network performance:

Table 5–14 Solaris 2.6 Network enhancements

Network Enhancement	Description
Kernel Sockets	Enables higher socket performance in addition to the TCP / IP STREAMS
TCP Large Windows	Allows TCP sessions to transmit larger packet sizes between 64K-1G
Zero Copy TCP / Hardware Checksum	Avoids data copy, and uses hardware checksum logic

Index

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W